

Arnold Willemer

Galileo Computing

Wie werde ich UNIX-Guru?





Arnold Willemer

Galileo Computing

Wie werde ich UNIX-Guru?

Inhalt

Vorwort 17

1	UNIX anwenden 23
1.1	Besonderheiten von UNIX 23
1.1.1	Das offene System 23
1.1.2	Die UNIX-Varianten 24
1.2	Personenkontrolle 26
1.3	Fragen Sie Dr. UNIX 28
1.3.1	Unterteilung 29
1.3.2	Sonderzeichen in Hilfetexten 30
1.4	So sage ich es meinem UNIX 31
1.5	Operationen mit Dateien 32
1.5.1	Eine kleine Beispielsitzung 33
1.5.2	Dateien anzeigen: ls 35
1.5.3	Dateien kopieren: cp 37
1.5.4	Dateien verschieben oder umbenennen: mv 38
1.5.5	Dateien löschen: rm 38
1.5.6	Verzeichnisbefehle: mkdir, rmdir, cd und pwd 38
1.6	Der UNIX-Verzeichnisbaum 39
1.6.1	Navigation 39
1.6.2	Ein Blick unter die Haube: i-nodes 40
1.6.3	Was ist wo? 41
1.6.4	Suche im Verzeichnisbaum: find 43
1.7	Dateieigenschaften 46
1.7.1	Eigentümer wechseln: chown und chgrp 47
1.7.2	Berechtigungen: chmod 48
1.7.3	Neuer Zeitstempel: touch 52
1.7.4	Links: Zwei Namen, eine Datei 52
1.7.5	Besondere Dateien 56
1.7.6	Der Dateityp: file 56
1.8	Zugriff auf mehrere Objekte 57
1.8.1	Wildcards: *, ? und die eckigen Klammern 57
1.8.2	Sonderzeichen als Parameter 58
1.9	Editoren 58
1.9.1	vi 59
1.9.2	emacs 69

1.10	UNIX-Kommandos verknüpfen	71
1.10.1	Ein- und Ausgabe als Datenstrom	72
1.10.2	Umleitung	72
1.10.3	Piping	74
1.10.4	Verschachtelte Befehlsargumente	75
1.11	Praktische Helfer	76
1.11.1	Ausgabe einer Datei: cat	76
1.11.2	Seitenweise: more	76
1.11.3	Durchsuchungsbefehl: grep	77
1.11.4	Wenn ich auf das Ende sehe: tail	78
1.11.5	Wortzähler: wc	78
1.11.6	sort	78
1.11.7	sed	79
1.11.8	awk	79
1.11.9	Weitere Werkzeuge im Überblick	84
1.12	Reguläre Ausdrücke	84
1.13	Pack deine Sachen und geh...	87
1.13.1	Verschnüren: tar	87
1.13.2	Zusammenpressen: compress und gzip	89
1.13.3	Kombination aus Packen und Pressen	89
1.14	Prozesse	90
1.14.1	Hintergrundbearbeitung und Reihenfolge	90
1.14.2	Prioritäten: nice	92
1.14.3	Ausloggen bei laufendem Prozess: nohup	93
1.14.4	Prozessliste anzeigen: ps	93
1.14.5	Stoppen eines Prozesses: kill	94
1.14.6	Programmabbruch	95
1.15	Umgebungsvariablen	96
1.16	Die Shell	98
1.16.1	alias	99
1.16.2	Startupdateien der Shell	100
1.16.3	Bourne-Shell (sh) und POSIX	100
1.16.4	Korn-Shell (ksh)	101
1.16.5	C-Shell (csh)	104
1.16.6	Bourne-Again-Shell (bash)	107
1.17	Ausgaben auf dem Drucker	108
1.17.1	BSD-Unix: lpr, lpq und lprm	109
1.17.2	AT&T: lp, lpstat und cancel	110
1.17.3	Druck formatieren: pr und a2ps	111
1.18	Zeitversetztes Arbeiten	111
1.18.1	Die aktuelle Zeit	112
1.18.2	Regelmäßige Arbeiten: crontab	113

- 1.18.3 Zeitversetzter Job: at 114
- 1.19 Diskettenlaufwerke 115**
 - 1.19.1 Formatieren und Beschreiben 115
 - 1.19.2 mount und eject 116
 - 1.19.3 tar und sync 116
 - 1.19.4 MS-DOS-Disketten 116
- 1.20 CD-ROMs 118**
- 1.21 CD-Brenner 118**
 - 1.21.1 Datensicherung 119
 - 1.21.2 RW-Medien 121
 - 1.21.3 Multisession 121
 - 1.21.4 Audio-CDs 122
- 1.22 Notebooks 124**
 - 1.22.1 PCMCIA 125
 - 1.22.2 Problematische Peripherie 126
 - 1.22.3 Software für den Akku 126

2 Administration 129

- 2.1 Die Arbeitsumgebung des Administrators 133**
- 2.2 Administrationstools 134**
- 2.3 Start des Systems 136**
 - 2.3.1 Bootprompt 137
 - 2.3.2 Bootkonfiguration: lilo 138
 - 2.3.3 Durchlaufen der Runlevel 139
 - 2.3.4 BSD: /etc/rc 141
 - 2.3.5 System V: init.d 142
 - 2.3.6 Konfigurationsdateien 144
- 2.4 Herunterfahren: shutdown 145**
 - 2.4.1 Alles bereit zum Untergang? 147
 - 2.4.2 Wechsel in den Single-User-Modus 148
- 2.5 Benutzerverwaltung 149**
 - 2.5.1 Passwortverwaltung unter UNIX 150
 - 2.5.2 Die Benutzerdatei /etc/passwd 151
 - 2.5.3 Verborgene Passwörter: shadow 153
 - 2.5.4 Benutzerpflege automatisieren 154
 - 2.5.5 Grundeinstellungen: /etc/profile 155
 - 2.5.6 Verzeichnisprototyp: /etc/skel 157
 - 2.5.7 Gruppenverwaltung 158
 - 2.5.8 Benutzerüberwachung 159
 - 2.5.9 Kurzfristiger Benutzerwechsel: su 161
 - 2.5.10 Administrationsaufgaben starten: sudo 162

- 2.5.11 Pseudobnutzer zum Shutdown 164
- 2.6 Hardwarezugriff unter UNIX: /dev 165**
 - 2.6.1 Aufgaben eines Treibers 166
 - 2.6.2 Gerätedateien 167
 - 2.6.3 Umgang mit Gerätedateien 169
 - 2.6.4 Gerätenamen 171
- 2.7 Festplatten 172**
 - 2.7.1 SCSI-Platten 174
 - 2.7.2 IDE-Platten 175
 - 2.7.3 Inbetriebnahme 176
 - 2.7.4 RAID-Systeme 177
 - 2.7.5 Partitionieren 181
 - 2.7.6 Dateisystem erstellen 183
 - 2.7.7 Swapping 184
 - 2.7.8 Einbinden eines Dateisystems: mount 186
 - 2.7.9 Konsistenz der Dateisysteme 189
 - 2.7.10 Journal-Dateisysteme 190
 - 2.7.11 Belegungslisten: df und du 191
 - 2.7.12 Zuteilung des Plattenplatzes: quota 192
 - 2.7.13 Maximalwerte 194
- 2.8 Datensicherung 196**
 - 2.8.1 Vorüberlegungen 197
 - 2.8.2 Das Bandlaufwerk 199
 - 2.8.3 dump 200
 - 2.8.4 tar (tape archiver) 204
 - 2.8.5 cpio 208
 - 2.8.6 Medien kopieren: dd 211
 - 2.8.7 Andere Sicherungstools: AMANDA 212
 - 2.8.8 Beispiel für eine Sicherung auf CD-RW 215
- 2.9 Software installieren 217**
 - 2.9.1 make als Installationswerkzeug 218
 - 2.9.2 Solaris Packages 219
 - 2.9.3 HP-UX: SD-UX 220
 - 2.9.4 Red Hat Package Manager 222
- 2.10 Druckeradministration 224**
 - 2.10.1 Übersicht 226
 - 2.10.2 BSD-Unix: lpd, lpr, lpq und lprm 227
 - 2.10.3 Linux-PC als Druckserver 231
 - 2.10.4 System V: lpsched, lp, lpstat und cancel 234
 - 2.10.5 LPRng 238
 - 2.10.6 CUPS – Common UNIX Printing System 239

- 2.11 Terminals 241**
 - 2.11.1 Konfiguration der Terminals 242
 - 2.11.2 Die Terminalvariable TERM 244
 - 2.11.3 termcap 245
 - 2.11.4 terminfo 247
 - 2.11.5 Wenn das Terminal durcheinander ist 248
- 2.12 Anschluss eines Modems 249**
- 2.13 Tuning 250**
 - 2.13.1 Optimierung des Dateisystems 251
 - 2.13.2 Wissen, wo der Schuh drückt 254
- 2.14 Informationen sammeln 258**
 - 2.14.1 Versionsinformationen: uname 259
 - 2.14.2 Der syslog-Dämon und die messages-Datei 260
 - 2.14.3 Umgang mit großen Protokolldateien 262
 - 2.14.4 Briefe aus dem Nirvana 265
 - 2.14.5 Bootzeitpunkt und Systemlast: uptime 266
 - 2.14.6 Prozessbeobachter 267
 - 2.14.7 Nicht immer mit Tötungsabsicht: kill 271
 - 2.14.8 Offene Dateien 273
 - 2.14.9 Programmzusammenbrüche (Coredump) 275
 - 2.14.10 Systemabsturz (Kernel-Panic) 276
- 2.15 Der Kernel 277**
 - 2.15.1 Dynamische Bibliotheken 279
 - 2.15.2 Module 280

3 Netzwerk 282

- 3.1 Client-Server-Architekturen 286**
 - 3.1.1 Ethernet als Verkabelungsbeispiel 287
 - 3.1.2 Pseudoschnittstelle loopback 289
 - 3.1.3 Pakete in Paketen 290
- 3.2 TCP/IP, der Standard 291**
 - 3.2.1 Die TCP/IP-Nummer 292
 - 3.2.2 Das Prüftool ping 298
- 3.3 Routing: Verbindung mehrerer Netzwerke 300**
 - 3.3.1 Gateways 301
 - 3.3.2 Statische Festlegung einer Route 302
 - 3.3.3 Statisches Routing: Ein Beispiel 304
 - 3.3.4 Subnetze 309
 - 3.3.5 Dynamisches Routen 311
 - 3.3.6 CIDR – Classless Inter-Domain Routing 312

3.4	Namensauflösung	313
3.4.1	Der Host- und Domainname	314
3.4.2	Die Datei /etc/hosts	315
3.4.3	Die Datei /etc/services	316
3.4.4	Domain Name Service: DNS	318
3.4.5	Network Information Service: NIS	326
3.4.6	Netzgruppen: /etc/netgroup	330
3.5	Next Generation IPv6	331
3.6	Die Grundausstattung an Bordwerkzeug	333
3.6.1	ICMP und ping	334
3.6.2	Verbindung zwischen Prozessen: netstat	336
3.6.3	Anzeigen der Netzwerkadapter	337
3.6.4	Anzeigen der Routingtabelle	339
3.6.5	Routen verfolgen: traceroute	340
3.6.6	HP-UX: lanadmin	341
3.7	TCP/IP-Dienste	342
3.7.1	inet-Dämon	344
3.7.2	File Transfer Protocol (FTP)	345
3.7.3	Anonymer FTP-Server	348
3.7.4	TFTP, schnell und vertrauensvoll	349
3.7.5	Terminaldienst (telnet)	350
3.7.6	Die r-Kommandos	352
3.7.7	Wenn Sicherheit vorgeht: die ssh und scp	356
3.7.8	NFS – Network File System	360
3.7.9	Automatisches Mounten	364
3.8	Allgemeines zum Internet-Anschluss	367
3.9	Dynamische TCP/IP-Nummern (DHCP)	369
3.10	E-Mail	371
3.10.1	Format einer E-Mail	372
3.10.2	UNIX und Mail	374
3.10.3	SMTP (Simple Mail Transport Protocol)	375
3.10.4	Mailqueue	376
3.10.5	Verteilen der Post: sendmail -q	377
3.10.6	Weiterleiten der Post: aliases und forward	378
3.10.7	Lokale Mail lesen	379
3.10.8	POP3	381
3.10.9	IMAP	383
3.10.10	Post sammeln: fetchmail	385
3.10.11	Mail-Server und Domain	386
3.10.12	Erstes Beispiel: Interne Firmenmail	387
3.10.13	Zweites Beispiel: Anbindung an das Internet	388

- 3.11 Newsgroups 390**
 - 3.11.1 News lesen 392
 - 3.11.2 Installation des Newsservers inn 393
 - 3.11.3 Beispiel: Newsserver zur Projektverwaltung 396
 - 3.11.4 Gruppen anlegen 398
 - 3.11.5 Verbindung nach außen 400
 - 3.11.6 Newsgroups saugen 402
 - 3.11.7 NNTP-Protokollbeschreibung 404
- 3.12 Jeder Rechner ist ein eigener Webserver 407**
 - 3.12.1 Hypertext und HTML 409
 - 3.12.2 Start des Servers 413
 - 3.12.3 Die Konfigurationsdatei httpd.conf 414
 - 3.12.4 Privatadministration per .htaccess 417
 - 3.12.5 Kommunikation per HTTP 419
 - 3.12.6 Virtuelles Hosting 422
 - 3.12.7 CGI: Der Server schlägt zurück 423
- 3.13 Fremdgegangen – Andere Protokolle 426**
 - 3.13.1 Samba: UNIX im Windows-Netz 427
 - 3.13.2 Novell-Zugriffe 433
 - 3.13.3 Mac im Netz: netatalk 435
- 3.14 Firewall und Masquerading 436**
 - 3.14.1 Funktionsweise einer Firewall 437
 - 3.14.2 Masquerading 440
- 3.15 Proxy 441**

4 Das X-Window System 444

- 4.1 Grafische Oberfläche unter UNIX 447**
 - 4.1.1 X, Fenstermanager, Widget Sets und Desktop 449
 - 4.1.2 Der X-Server 450
 - 4.1.3 Der Fenstermanager 451
 - 4.1.4 Der X-Client und seine Bibliotheken 452
- 4.2 X Window starten 455**
 - 4.2.1 Nacktstart mit xinit 456
 - 4.2.2 Regulärer Start von X: startx 457
 - 4.2.3 Grafisches Einloggen: Display Manager xdm 458
 - 4.2.4 Eine kleine Beispielsitzung mit xdm 459
- 4.3 X Window benutzen 461**
 - 4.3.1 Bedienungselemente des Athena Widget Set 462
 - 4.3.2 Der Aufruf von X-Programmen 464
 - 4.3.3 Cut and Paste 466
 - 4.3.4 Terminalfenster xterm 467

4.3.5	Weitere praktische Helfer	470
4.4	Konfigurieren	471
4.4.1	Farbbeschreibung	472
4.4.2	Schriften	473
4.4.3	Bitmaps	475
4.4.4	Ressourcen	476
4.4.5	Konfiguration des Fenstermanagers	479
4.4.6	Fokus und Z-Anordnung	480
4.5	Desktops	481
4.5.1	CDE	483
4.5.2	KDE	488
4.5.3	GNOME	495
4.5.4	Der Wettstreit der freien Desktops	502
4.5.5	Mac OS X	503
4.6	Das X Window System im Netz	506
4.6.1	X-Programme über das Netz starten	507
4.6.2	X-Server-Software in Betrieb nehmen	509
4.6.3	Grafisches Einloggen über das Netz	510
4.6.4	Thin Client	514
4.7	Anwendersoftware für UNIX	515
5	Programmierung von Shellskripten	518
<hr/>		
5.1	Erstellen und Start eines Shellskripts	522
5.2	Variablen	523
5.3	Ablaufsteuerung	526
5.3.1	Die Unterscheidung: if	527
5.3.2	Bedingungen	528
5.3.3	Rückgabewert von Programmen	531
5.3.4	Die Fallunterscheidung: case	532
5.3.5	Die while-Schleife	534
5.3.6	Die for-Schleife	536
5.3.7	Funktionen	538
5.4	Ein- und Ausgaben aus dem Skript	539
5.5	Start und Umgebung von Skripten	540
6	Perl	542
<hr/>		
6.1	Interpreter und Skript	544
6.2	Variablen	545
6.2.1	Skalare	546

6.2.2	Variablennamen	548
6.2.3	Operationen auf Skalare	550
6.2.4	Arrays	552
6.2.5	Hash	554
6.3	Interaktiv	555
6.3.1	Ein- und Ausgabe	556
6.3.2	Aufrufparameter	557
6.3.3	Umgebungsvariablen	558
6.4	Ablaufsteuerung	559
6.4.1	Bedingungen	560
6.4.2	if	562
6.4.3	for	563
6.4.4	foreach	565
6.4.5	Sonstige Schleifen: while und until	566
6.4.6	Funktionen	568
6.5	Dateien	569
6.5.1	Schreiben und Lesen	570
6.5.2	Umgang mit Dateien	572
6.6	Perl und UNIX	573
6.6.1	Aufruf von UNIX-Programmen	574
6.6.2	UNIX-Systemprogrammierung	575
6.7	Grafische Oberfläche: Tk	576
6.7.1	Widgets und Ressourcen	578
6.7.2	Kontrollelemente	579
6.7.3	Widgetanordnung	587
6.8	Informationsquellen	590
7	Programmierwerkzeuge	591
<hr/>		
7.1	C-Compiler	592
7.2	make	594
7.3	Debugger	599
7.3.1	dbx	601
7.3.2	adb (System V)	602
7.3.3	gdb GNU debug	603
7.4	Analysewerkzeuge	605
7.4.1	Systemaufrufe verfolgen: strace und ltrace	606
7.4.2	Speicherlecks und -überläufe	607
7.5	Versionsverwaltung	609
7.5.1	SCCS (Source Code Control System)	610
7.5.2	RCS (Revision Control System)	611

- 7.5.3 Zusammenspiel mit make 612
- 7.5.4 CVS (Concurrent Versions System) 613
- 7.5.5 UNIX als CVS-Server 616
- 7.6 Diverse Programmierhelfer 618**
- 7.6.1 Kurzbetrachtung: lex und yacc 619
- 7.6.2 Unterschiede zwischen Textdateien: diff 620
- 7.6.3 Dateien aufs Byte geschaut 621

8 UNIX-Systemaufrufe 622

- 8.1 Die Funktion main 624**
- 8.1.1 Aufrufparameter 625
- 8.1.2 Zugriff auf die Umgebungsvariablen 627
- 8.2 Fehlerbehandlung: errno 629**
- 8.3 Dateizugriffe 630**
- 8.3.1 Öffnen, Lesen und Schreiben 632
- 8.3.2 Positionieren: lseek 635
- 8.3.3 Dateihandle duplizieren: dup 636
- 8.3.4 Datei-Eigenschaften ermitteln 637
- 8.3.5 Datei-Eigenschaften ändern 640
- 8.3.6 Sperren 642
- 8.3.7 Link erzeugen: link, symlink 647
- 8.3.8 Löschen: unlink 648
- 8.3.9 Umbenennen: rename 649
- 8.3.10 Temporäre Dateien 650
- 8.4 Verzeichnisse 651**
- 8.4.1 Auslesen: opendir, readdir, closedir 652
- 8.4.2 Ermitteln des Arbeitsverzeichnisses 654
- 8.4.3 Wechseln: chdir 655
- 8.4.4 Anlegen und Löschen: mkdir, rmdir 656
- 8.5 Prozesse 657**
- 8.5.1 Multiprocessing contra Multithreading 659
- 8.5.2 Vervielfältigen von Prozessen: fork 660
- 8.5.3 exec und system 662
- 8.5.4 Synchronisation: wait 663
- 8.5.5 Prozessumgebung 664
- 8.5.6 Gemeinsamer Speicher: Shared Memory 666
- 8.5.7 Synchronisation mit Semaphoren 670
- 8.5.8 Message Queues 673
- 8.5.9 Leichtgewichtsprozesse: Threads 677
- 8.6 Signale 680**
- 8.6.1 Signale senden: kill 682

- 8.6.2 Auf Signale warten: pause 683
- 8.6.3 Timeout setzen: alarm 684
- 8.6.4 Zombies vereiteln 685
- 8.7 Pipe 686**
 - 8.7.1 Prozesskommunikation per Pipe 687
 - 8.7.2 Named Pipe oder FIFO 688
 - 8.7.3 Drucken unter UNIX 689
- 8.8 Fehlerbehandlung mit syslog 690**
- 8.9 Zeitfunktionen 692**
- 8.10 Benutzer und Gruppen 694**
 - 8.10.1 Die Passwortdatei als Struktur 695
 - 8.10.2 Auslesen der Passwortdatei 696
 - 8.10.3 Gruppen 697
- 8.11 Grundlagen der Dämonisierung 699**
- 8.12 Client-Server-Socketprogrammierung 700**
 - 8.12.1 Kommunikationsendpunkt: socket und close 702
 - 8.12.2 Serveraufrufe: bind, listen und accept 703
 - 8.12.3 Clientaufruf: connect 704
 - 8.12.4 Datenaustausch: send und recv 705
 - 8.12.5 Namensauflösung 706
 - 8.12.6 Zahlendreher ntoh und hton 707
 - 8.12.7 Rahmenprogramm eines Client-Server-Paars 708
 - 8.12.8 Mehrere Sockets parallel abfragen 712
 - 8.12.9 IPv6 aus Programmiersicht 714
 - 8.12.10 Client-Server aus Sicht der Performance 715
- 8.13 Reguläre Ausdrücke 716**
- 8.14 Weitere Programmierschnittstellen 718**
- 8.15 Systemkonformität 719**
 - 8.15.1 Polling 720
 - 8.15.2 Rechte beachten 721

A Glossar 722

B Literatur 723

Vorwort

Dieses Buch wurde für den Einsteiger geschrieben. Leicht verständlich sollte es sein. Da ich mich oft darüber ärgerte, dass ich von Einsteigerbüchern immer so bald im Regen stehen gelassen wurde, sollte dieses Buch so lange wie möglich nützlich sein. Also sollten zu möglichst vielen Stichworten, die einem Anfänger begegnen können, ein paar erklärende Worte fallen. Na ja, dann kann man ja gerade noch ein paar Worte dazu sagen, wie man das installiert und konfiguriert.

Motivation

Was sollte in das Buch hinein? Klar, die grundlegenden Befehle zum Umgang mit Dateien und Verzeichnissen sind wichtig. Die Editoren vi und emacs und die Shell sind die Standardwerkzeuge eines jeden UNIX-Anwenders und müssen natürlich erläutert werden. Neben diesen klassischen Themen will man aber auch mit einem CD-Brenner arbeiten. Auch die Administration darf nicht fehlen. Immerhin gibt es immer mehr Einzelplatzbenutzer, insbesondere im Bereich Linux und Mac OS X, die letztlich ihr System auch selbst administrieren müssen. Wer es im professionellen Umfeld mit UNIX zu tun bekommt, wird schnell in das Umfeld der Administration geraten. Und dann wird man über kurz oder lang auch mal ein paar kleine Skripte zaubern müssen. Wenn man dann als Administrator Perl kennt, ist das nicht verkehrt. Auf der anderen Seite kann man in der heutigen Zeit das Thema grafische Oberflächen nicht mehr weglassen. Und dass eine Maschine ohne Netzwerk betrieben wird, ist im Internet-Zeitalter einfach unglaubwürdig. So kam dann eins zum anderen. Inzwischen ist dieses Buch fast schon eine UNIX-Enzyklopädie, aber immer noch soll es für den Anfänger leicht verständlich sein. Denn Verständlichkeit schätzt nicht nur der Anfänger, sondern auch der Profi, wenn ihm mal ein neues Thema begegnet.

Natürlich wird man nicht durch die Lektüre dieses Buchs allein zum UNIX-Guru. Das wird auch sicher nicht wirklich jemand vermuten. Guru wird man, wenn man sich gern und intensiv immer wieder mit UNIX befasst. Aber ich bin doch sicher, dass dieses Buch auf dem Weg dorthin immer wieder eine Hilfe darstellt und die Richtung weist. Und so hoffe ich, dass Sie dieses Buch nicht im Regen stehen lässt.

UNIX ist ein feines Betriebssystem. Vor allem ist es ein offenes System. Das bedeutet, dass man alles über UNIX erfahren kann, wenn man es will. Aber man muss sich etwas damit befassen. Denn Dilettantismus fliegt einem zu, Wissen nicht. Aber entgegen aller Klischees über UNIX ist es eigentlich kein wirklich kompliziertes System. Ich halte MS Windows für

Was ist UNIX?

wesentlich komplizierter. Aber das wird Bill Gates wahrscheinlich anders sehen. UNIX ist sicher, stabil und leistungsfähig, und darum ist es sinnvoll, sich damit zu befassen.

Linux und UNIX Inzwischen besteht kein ernsthafter Zweifel mehr daran, dass Linux ein »echtes« UNIX ist. UNIX verdankt seine derzeitige Popularität vor allem Linux und selbst denjenigen, die sich noch mit Wehmut an die Tage der Exklusivität ihres Expertentums zurückerinnern, ist inzwischen klar, dass UNIX ohne Linux heute vielleicht nur noch eine Fußnote der EDV-Geschichte wäre. Da Linux in einem wesentlich schnelleren Tempo entwickelt wird, kann man unter Linux bereits vorab sehen, was in den anderen UNIX-Varianten später einmal Standard sein wird.

Mac OS X und UNIX Auch Apple ist mit Mac OS X zu einem wichtigen Mitspieler in der UNIX-Arena geworden. Das bislang proprietäre Betriebssystem basiert nun auf FreeBSD. Zu dem Darwin genannten Kernel, den es als Open Source gibt, kommen eine eigene grafische Benutzeroberfläche namens Aqua sowie entsprechende Programmierschnittstellen (Carbon und Cocoa) hinzu, auf die dieses Buch allerdings nicht näher eingehen wird. Wenn Sie also etwas über den neuen Kernel von Mac OS X erfahren wollen, sind Sie hier richtig. Norbert M. Doerner, der als Autor von CDFinder auf dem Macintosh bekannt ist, hat beim Korrekturlesen immer wieder darauf hingewiesen, welche Eigenheiten das Mac OS X hat, und natürlich sind seine Anmerkungen in dieses Buch eingeflossen.

- ▶ **Erstes Kapitel: Anwendung** Hier steht die Anwendung und der Einstieg in UNIX im Mittelpunkt. Da die Kommandosprache bei UNIX extrem leistungsfähig ist, ist es sinnvoll, sie zu kennen, selbst wenn man in erster Linie die grafische Oberfläche benutzt.
- ▶ **Zweites Kapitel: Administration** Die klassischen Administrationstätigkeiten wie Benutzerverwaltung und Datensicherung werden hier genauso erläutert wie grundlegende Ansätze zur Analyse von Systemproblemen.
- ▶ **Drittes Kapitel: Netzwerk** UNIX und TCP/IP gehören zusammen. Sowohl der Bereich der lokalen Netzwerke als auch die Dienste im Internet werden hier beschrieben.
- ▶ **Viertes Kapitel: X Window System** Heutzutage wird UNIX nur noch in reinen Serverumgebungen ohne grafische Oberflächen verwendet. Neben einer kurzen Beschreibung der Benutzung und der Möglichkeiten der Desktops CDE, KDE und GNOME wird hier die Konfiguration bis hin zum grafischen Einloggen über das Netz beschrieben.

- **Fünftes Kapitel: Shellskripten** Das Erstellen kleiner Skripten für Alltagsaufgaben ist unter UNIX sehr einfach. Die Skriptsprache ist aber so mächtig, dass auch komplexere Abläufe damit programmiert werden können. Wer einen Einblick in die Grundlagen der Programmierung sucht, findet in den Shellskripten eine leicht erreichbare Umgebung.
- **Sechstes Kapitel: Perl** Da diese Skriptsprache sowohl für Administration als auch im Internet-Bereich häufig verwendet wird, lohnt sich die Einarbeitung.
- **Siebttes Kapitel: Programmierwerkzeuge** Unter UNIX gibt es eine große Zahl hilfreicher Werkzeuge. Hier werden Compiler, make, einige Versionsverwaltungen und andere hilfreiche Werkzeuge beschrieben.
- **Achstes Kapitel: Systemaufrufe** Wer Software für UNIX erstellen will oder auch nur einen tieferen Einblick in den Umgang mit Prozessen und Dateien bekommen möchte, sollte sich die Systemaufrufe anschauen.
- **Glossar** Hier findet vor allem der Anfänger Begriffe erläutert, die im Text vielleicht etwas zu kurz kommen, um den Lesefluss nicht zu stören.

Zur besseren Lesbarkeit sind folgende Konventionen eingeführt worden: Aufrufe und Kommandos werden in nichtproportionaler Schrift gesetzt. **Dateien** und **Verzeichnisse** erscheinen in fetter Schrift. Funktionsaufrufe wie `open()` sind grundsätzlich mit Klammern dargestellt. KONSTANTEN und Umgebungsvariablen sind unter UNIX meist in Großbuchstaben gesetzt. Da dies bereits leicht aus dem Schriftbild heraussticht, habe ich es dabei belassen.

Schreibweisen

Dieses Symbol am Rand soll darauf hinweisen, dass hier ein Beispielszenario aufgezeigt wird, das im Text weiter verfolgt wird. Da viele Beispiele im Buch verwendet werden, sind nicht alle so auffällig gekennzeichnet, sondern nur solche, die etwas ausführlicher behandelt werden.



Mit diesem Symbol werden Hinweise gekennzeichnet, wie Sie sich das Leben etwas erleichtern.



Dieses Symbol soll auf Stolperfallen hinweisen. Hier schleichen sich entweder leicht Fehler ein, oder es geht um Dinge, die zu einem Datenverlust oder zum Verlust der Systemsicherheit führen können.



Bei diversen Beispielen werden Bildschirmabzüge dargestellt. Dabei zeigt der Prompt immer den Rechnernamen und ein Größerzeichen für einen normalen Anwender und ein Hashzeichen (#) für den Administrator root.

Konsolenprompt

Hier stehen sie untereinander:

gaston >
silver #

Dabei sind die Rechnernamen bei mir etwas bunt. Mein Linux-Arbeitsplatz heißt gaston, dann gibt es noch silver (Linux), powermac (Mac OS), hpsrv (HP-UX), note (FreeBSD), sol (Solaris), sparc (SunOS) und andere.

Herzlichen Dank Wenn man ein Buch schreibt, arbeitet man nicht im luftleeren Raum. Da gibt es ein paar Leute, die mich unterstützt haben, und das war sehr nett. Meine Frau Andrea und meine Söhne haben mir einen Freiraum gewährt, in dem ich arbeiten konnte. Dankenswerterweise wurde ich mit Nahrungsmitteln versorgt, sodass ich bei Abschluss der Arbeiten sogar noch mehr wiege als vorher. Stephan Mattescheck hat mich als Lektor betreut, mir ständig Bücher zugeschickt, Fragen gestellt und auch manche beantwortet. Er hat mich weitgehend walten lassen und sich als Partner angeboten, wenn ich mir unsicher wurde. Daniel Lauer hat mich im Layout und bei meinen Problemen mit den Untiefen von \LaTeX unterstützt, Iris Warkus hat mir einige Geheimnisse aus dem Bereich der Grafik verraten und ist verantwortlich für das gute Aussehen des Buches. Frau Friederike Daenecke formte aus meinen Verbrechen gegen die deutsche Sprache druck- und lesbare Sätze. Die Firma Tacoss hat mich mit Unterlagen unterstützt, mir freien Zugang zu ihrem Maschinenpark gewährt und mir eine HP-UX-Maschine zur Verfügung gestellt. Insbesondere Claus Erichsen und Leif Hansen seien hier mit Namen genannt. Sehr viel Arbeit haben sich diejenigen gemacht, die dieses Buch zur Korrektur gelesen und viele Anregungen eingebracht haben. Das ist Ralf Lenz mit seinen Erfahrungen, die er bei diversen Internet-Providern als Programmierer, Projektleiter und Netzwerkexperte gesammelt hat. Norbert M. Doerner hat seine Administrationserfahrungen aus dem Umfeld von Solaris und seine Kenntnisse als Entwickler auf dem Macintosh eingebracht. Er hat auch die Informationen dieses Buches darauf geprüft, ob sie für Mac OS X gelten, und Hinweise auf Unterschiede gegeben. Jörg Osarek von der Firma Oracle hat mit seinen UNIX- und Linux-Kenntnissen aus der Perspektive des professionellen EDV-Einsatzes wichtige Aspekte einbringen können und trotz engem Terminkalender noch Zeit für dieses Buch gefunden. Stephan Engelmann hat mit seiner Erfahrung als Webadministrator und Netzwerkpraktiker wichtige Fragen gestellt und gute Hinweise gegeben.

Norgaardholz, den 28.9.2002
Arnold Willemer

Teil 1

Anwendung

1 UNIX anwenden

UNIX ist ein benutzerfreundliches System. Es ist nur manchmal etwas eigen in der Auswahl seiner Freunde.

Dieser Satz charakterisiert UNIX vielleicht besonders. Es ist tatsächlich benutzerfreundlich und nicht so schwer zu lernen. Aber man muss etwas mehr tun, als mit der Maus wahllos auf Bildchen zu klicken. Man lernt UNIX nicht durch Trial and Error (also Versuch und Irrtum), sondern dadurch, dass man die angebotenen Hilfen nutzt, um zu verstehen, was passiert.

Das erste Kapitel soll zeigen, wie man sich unter UNIX auf der Konsole bewegt, und die einfachsten Grundkommandos vermitteln. Einige Fachbegriffe sind im Glossar im Anhang erläutert.

In Zeiten der grafischen Oberflächen muss man nicht zwingend mit der textorientierten Oberfläche arbeiten, auch unter UNIX nicht. Aber gerade unter UNIX wäre es ein Verlust, sie nicht zu kennen. Die Kommandozeile eröffnet Möglichkeiten zur Kombination der sehr leistungsfähigen UNIX-Werkzeuge, die in dieser Form auf grafischen Oberflächen nicht zu realisieren ist. So drückt die Verwendung der Kommandozeile, die die meisten UNIX-Anwender nach wie vor neben der grafischen Oberfläche nutzen, nicht aus, dass die grafischen Oberflächen nicht brauchbar seien, sondern dass die Kommandozeile extrem leistungstark ist.

Die Konsole

Aber ein Textbildschirm hat noch andere Vorteile. Für den Administrator ist es wichtig, dass er selbst über die langsamste Telefonleitung noch eine Fernwartung durchführen kann.

1.1 Besonderheiten von UNIX

1.1.1 Das offene System

Das Besondere an UNIX ist seine Offenheit. UNIX ist nach seiner Entstehung lange Zeit an der Berkeley-Universität weiterentwickelt worden. Der Sourcecode stand Studenten zur Verfügung, damit sie lernen, wie ein Betriebssystem funktioniert. Vieles wurde an UNIX demonstriert, und was UNIX nicht konnte, das brachte man ihm bei. Vor allem aber sorgte man dafür, dass es keine Geheimnisse gab. Jeder Prozess ist sichtbar, und man kann ihm auf die Finger klopfen. Alle Konfigurationsdateien sind reine Textdateien oder leicht in solche zu verwandeln. Solche Konfigurationen

kann man leicht sichern, ausdrucken oder durchsuchen. Eine solche Umgebung ist aber auch sicher, weil einfach nichts versteckt werden kann.

**Komplex, aber
nicht kompliziert**

UNIX gilt als kompliziert. Dieses Klischee wird von Leuten verbreitet, die glauben, MS Windows sei simpel. Ein modernes Betriebssystem mit Multitasking, Netzwerkanschluss und grafischer Oberfläche ist komplex. Wer den Zugriff auf alle Details erhält, wie das bei UNIX der Fall ist, der mag zu Anfang über die vielen Informationen erschrecken. Aber man muss nicht alles wissen, um mit UNIX produktiv umgehen zu können. Dennoch ist es gut zu wissen, dass man alles wissen könnte.

Auf der Konsole arbeitet UNIX mit sehr kleinen Programmen, die wie Legosteine zusammengesetzt werden können. Lassen Sie sich davon faszinieren, welche Leistung man mit ein paar Tastendrücken entfachen kann. Denn was man mag, das lernt man schnell.

1.1.2 Die UNIX-Varianten

Hier wird nicht die komplette Geschichte von UNIX zelebriert, da der Nutzen gering und der Wahrheitsgehalt der umgehenden Legenden schwer prüfbar ist. Um aber die feinen Unterschiede zu begreifen, die zwischen den UNIX-Derivaten existieren, kommt man um einen kurzen Rückblick nicht herum.¹

Portables System In den 70ern entstand UNIX bei AT&T. Als erstes System wurde es nicht in der Assemblersprache des Rechners, sondern in einer extra für diesen Zweck entwickelten portablen Hochsprache entwickelt, die man heute unter dem Namen C kennt. Dadurch ergab sich die Möglichkeit, UNIX auf beinahe beliebige Plattformen zu portieren.

UNIX an der Uni Die kommerziellen Möglichkeiten wurden zunächst völlig unterschätzt, sodass man eine Lizenz an die Universität Berkeley vergab. Diese Lizenz umfasste auch den Sourcecode von UNIX, sodass Generationen von Studenten anhand von UNIX lernten, wie ein Betriebssystem funktioniert, und Erweiterungen bzw. Korrekturen vornahmen. Es entstanden die BSD-Versionen von UNIX, deren aktuelle Version 4.4BSD heißt. FreeBSD ist eine freie Variante nicht nur für die PC-Architektur.

System V Parallel dazu entwickelte AT&T System V.² Diese Variante ist die Basis von AIX von IBM, von HP-UX von Hewlett Packard und auch der neuesten

1 vgl. Herold, Helmut: Linux-UNIX-Systemprogrammierung. Addison-Wesley, 1999. S. 35–37.

2 Das V steht für eine römische Fünf und wird üblicherweise englisch five ausgesprochen.

Solaris-Versionen von Sun. Aufgrund der Notwendigkeit, die Portabilität zu wahren, die zu den ursprünglichen Stärken von UNIX gehörte, wurden auch Besonderheiten des BSD in das System V aufgenommen.

Während seiner Entwicklungsgeschichte drohte immer wieder die Zersplitterung von UNIX in diverse Dialekte. Um verbindliche Normen zu schaffen, gab es mehrere Bestrebungen. Das IEEE (Institute for Electrical and Electronic Engineers) definierte mit POSIX (Portable Operating System Interface) eine Familie von Standards für die UNIX-Schnittstellen.

POSIX von IEEE

Von Seiten der großen Computerhersteller wurde die X/Open gegründet, die einen Industriestandard für offene Systeme schaffen sollte. Das wichtigste Ergebnis war der XPG (X/Open Portability Guide).

X/Open

Nachdem UNIX aufgrund der kommerziellen Nutzung den Universitäten nicht mehr im Quelltext zur Verfügung stand und Lizenzen für interessierte Studenten unerschwinglich waren, ergaben sich verschiedene Ansätze, um in dieser Situation Abhilfe zu schaffen. An den Universitäten entstand zunächst XINU³ von Douglas Comer als ein Systemkern ohne Anbindung an irgendwelche Peripherie. Andrew Tanenbaum entwickelte MINIX, um daran grundlegende Betriebssystemeigenschaften zu demonstrieren.⁴ Der Ansatz war durch sein durchgängiges Konzept des Message Passing elegant, aber nicht auf Performance ausgelegt. Parallel arbeitete eine Gruppe von Leuten bereits an freier Software, die neben vielen Tools einen freien Compiler umfasste, den GNU-Compiler⁵.

Die Suche nach einem UNIX für Studenten

In dieser Situation erstellte Linus Torvalds einen Kernel in UNIX-Machart und stellte ihn im Internet zur freien Verfügung. Er nannte ihn Linux. Durch das Hinzufügen der GNU-Tools und des MINIX-Dateisystems konnte man daraus schnell ein lauffähiges Grundsystem bauen. Von FreeBSD kamen die ersten Netzbibliothek, und X Window wurde, da es frei verfügbar war, auch bald eingebunden. Anfangs wurde Linux noch als Betriebssystemspielzeug belächelt. Inzwischen hat sich durch die weltweite Beteiligung in unglaublicher Geschwindigkeit ein System entwickelt, das seine Spuren auch in anderen UNIX-Systemen hinterlässt. Wollte früher Linux wie UNIX sein, versuchen heute die UNIX-Systeme, die neuen Möglichkeiten von Linux zu integrieren. Heute gibt es wohl niemanden mehr, der daran zweifelt, dass Linux ein »richtiges« UNIX ist.

Linux

3 Comer, Douglas: Operating System Design – The XINU Approach. Prentice Hall International Editions, 1984.

4 Tanenbaum, Andrew S.: Operating Systems – Design and Implementation. Prentice Hall, 1987.

5 Die Abkürzung GNU steht für »GNU is Not Unix«. <http://www.gnu.org>

1.2 Personenkontrolle

Die erste Kontaktaufnahme mit einer UNIX-Maschine erfolgt normalerweise durch die Anmeldung am System, den so genannten Login. UNIX ist ein Multiuser-Betriebssystem. Es ist also darauf ausgelegt, für mehrere Benutzer eigene Arbeitsumgebungen zu schaffen und diese auch gegen bössartige Attacken zu schützen. Darum gibt es einen Benutzernamen und ein Passwort.

Benutzername Ein Terminal (siehe S. 638) meldet sich typischerweise nach dem Drücken der Returnntaste mit der Frage nach dem Benutzernamen. Auf den meisten Systemen wird der Administrator eine Kombination aus Vor- und Nachnamen für den Benutzernamen wählen. UNIX unterscheidet zwischen Groß- und Kleinschreibung. Darum verwendet man ungern Großbuchstaben, und so ist auch meist der Benutzernamen kleingeschrieben.

Passwort Nach dem Return verlangt das System das Passwort, das beim Eintippen nicht sichtbar ist. Wenn man sich vertippt hat, erscheint der Anmeldebildschirm erneut, ansonsten grüßt das System und meldet, wie viele erfolglose Anmeldeversuche seit dem letzten Login erfolgt sind. Dies ist eine wichtige Kontrolle, ob jemand vielleicht illegalerweise versucht hat, in das System einzubrechen. In solch einem Fall sollte man den Administrator informieren.

Grafischer Login Wenn Sie sich über einen grafischen Login angemeldet haben, erscheint zunächst eine grafische Oberfläche. Dort ist vielleicht kein Fenster mit einer Terminalemulation offen. Sie werden aber sicher leicht eine solche finden. Suchen Sie nach einem Bild, das ein Terminal oder einen Bildschirm darstellt. Oder schauen Sie, ob im Menü xterm auftaucht. Mit dieser Terminalemulation können Sie alles nachvollziehen, was in diesem Buch über Kommandozeilen beschrieben wird. Im Kapitel über das X Window System ist unter dem Titel »Eine kleine Beispielsitzung mit xdm« für die verschiedenen Systeme beschrieben, wie man in einfachen Schritten an eine solche Terminalemulation gelangt. Wenn Sie eine eher ältere UNIX-Workstation verwenden, finden Sie auf Seite 421 diese Beschreibung. Verwenden Sie CDE, das auf Solaris bis Version 8, auf Hewlett Packard und anderen RISC-Maschinen läuft, ist die Beschreibung auf Seite 442 zu finden. Die Beschreibung für GNOME (Linux und Solaris ab Version 9) findet sich auf Seite 459. Benutzer des KDE (auch Linux) finden Hinweise auf Seite 451. Auch Besitzer des Macintosh ab Mac OS X können mitmachen. Ab Seite 464 ist beschreiben, wie man an eine Kommandozeile gelangt.

Wenn man eine neue Kennung an einem System bekommen hat, sollte man das Passwort gleich nach dem ersten Einloggen ändern. Dazu gibt man einfach den Befehl `passwd` ein und schließt die Zeile mit Return. Die meisten Systeme fragen nach dem bisherigen Passwort. Auf diese Weise wird verhindert, dass jemand eine kurze Abwesenheit nutzt, um schnell das Passwort zu ändern. Anschließend müssen Sie zweimal Ihr neues Passwort eingeben. In diesem Fall werden Sie Ihre Eingabe nicht sehen können. Das verhindert, dass jemand Ihr Passwort vom Bildschirm abliest.

Passwort ändern

Sollte Ihr Passwort zentral in einem UNIX-Netzwerk verwaltet werden, kann es sein, dass der Befehl zum Ändern des Passwortes `yppasswd` lautet. In diesem Fall wird Ihnen der Administrator aber vermutlich schon einen Tipp gegeben haben.

**Netzwerk-
passwort**

Bei der Wahl des Passwortes sollten Sie alles vermeiden, was man leicht mit Ihnen verbindet. Namen von Verwandten sind ebenso unglücklich wie Ihr Autokennzeichen oder Ihr Sportverein. Auch Begriffe, die man in einem Lexikon findet, sollten Sie meiden. Eine einfache Methode, ein Passwort sicher zu machen, ist eine Ziffer irgendwo einzufügen. Auch die Verwendung von wild gemischter Groß- und Kleinschreibung steigert den Wert. Aber wählen Sie es so, dass Sie es sich merken können. Wenn Sie das Passwort auf einem Zettel in der Schublade aufbewahren müssen, weil es so kompliziert ist, dann ist es wertlos. Es gibt ein einfaches Verfahren, ein Passwort zu erzeugen, das aus scheinbar sinnlosen Buchstabenkolonnen besteht und dennoch leicht zu merken ist. Dazu bildet man einen Satz und verwendet die Anfangsbuchstaben als Passwort. Beispielsweise würde der Satz »Wie werde ich UNIX-Guru?« das Kennwort `WwiUG` ergeben.

**Ein gutes
Passwort**

Nach der Arbeit mit UNIX meldet man sich wieder ab. Man sollte sich zur Regel machen, niemals ein angemeldetes Terminal allein stehen zu lassen. Die Anmeldung kann durch Drücken von `ctrl-D`⁶ oder den Befehl `exit` oder `logout` erfolgen. Der Befehl `exit` und die Tastenkombination `ctrl-D` führen beide zum Verlassen der aktuellen Shell. Sofern keine weitere Shell eröffnet wurde, führt das wie der Aufruf `logout` zum Abmelden.

**Abmelden am
Terminal**

Bei einer grafischen Oberfläche finden Sie die Möglichkeit, sich abzumelden, meist über ein Menü, das Sie mit der rechten (oder linken) Maustaste

⁶ `ctrl` meint die Control-Taste, die auf deutschen Tastaturen meist mit `Strg` (Steuerung) beschriftet ist. Die Control-Taste wird gedrückt gehalten, während kurz eine andere Taste gleichzeitig gedrückt wird.

auf dem Desktophintergrund aufrufen. Oft ist auch ein X-Symbol zu finden, das zum Abmelden dient.

**Der Superuser
root**

Neben den normalen Anwendern eines Systems, gibt es den Systemadministrator, der unter UNIX traditionsgemäß den Benutzernamen root hat. Gegenüber den normalen Anwendern hat root freien Zugriff auf alle Ressourcen der Maschine. Aus diesem Grund wird er auch Superuser genannt. Für alle Systemkonfigurationen muss er angesprochen werden. Aus Sicherheitsgründen hat der normale Anwender eingeschränkte Rechte, die verhindern sollen, dass er einen anderen Benutzer schädigen oder ausspähen kann.

1.3 Fragen Sie Dr. UNIX

man, xman

Unter UNIX bekommt man die Informationen zum System mitgeliefert. Die erste Informationsquelle sind die so genannten Manpages. Sie sind benannt nach dem Kommando man (unter grafischer Oberfläche auch xman) und den Seiten (engl. pages), die dieser Befehl anzeigt. Der Name des Befehls man leitet sich von manual (engl. Handbuch) her. Man tippt hinter dem Kommando man ein Leerzeichen und dann den Befehl oder die Datei, über die man Informationen braucht. Man bestätigt mit der Return-Taste und es erscheinen klar strukturierte Informationsseiten, die üblicherweise sehr üppig sind. Zugegebenermaßen sind sie nicht für Anfänger geschrieben worden, sondern sollen eine klare Definition der beschriebenen Kommandos und Dateien liefern. Ein guter Einstieg in die Benutzung von man ist naheliegenderweise die Manpage von man. Man ruft sie mit man man auf. Mit der Leertaste kann man seitenweise weiterblättern, und mit Q kommt man wieder zur Kommandozeile zurück. In vielen Systemen kann man mit der Taste B (backwards) seitenweise zurückblättern. Selbst die Cursor-Tasten funktionieren manchmal.

**Nach Themen
suchen**

Mit der Option -k wird eine Liste aller Kommandos oder Dateien aufgeführt, die zu dem angegebenen Schlüsselwort passen. Beispielsweise zeigt man -k sockets einige Seiten zum Thema Socketprogrammierung an.

Gibt es doch keine Manpage oder will man nur einen kurzen Hinweis zur Art des Aufrufs, kann man das Kommando selbst mit der Option -? oder --help (bei Worten als Option braucht man zwei Bindestriche) aufrufen.

info

Unter einigen UNIX-Systemen (AIX, Linux) gibt es das Programm info, mit dem man sich durch einen Baum von Tutorien hangeln kann. Wer Zugriff auf die so genannten Linux HowTos hat, sollte sie nutzen. Da Linux auch ein UNIX ist, stellen sie auch für andere UNIX-Systeme ausgiebige

Hilfen dar. Man findet die HowTos und andere Dokumente auf einem Linux-System unter den Verzeichnissen `/usr/doc` oder unter `/usr/share/doc`.

Die Quelle für die Linux Howtos ist:

<http://www.linuxdoc.org>

Eine wichtige Quelle sind die Newsgroups. Hier findet man Hilfe, wenn man ein bestimmtes Problem nicht lösen kann. Gerade im UNIX-Bereich gibt es eine ganze Reihe solcher Gruppen, in denen man mit sehr kompetenten Partnern über seine Probleme diskutieren kann. Allerdings erwarten diese Experten auch, dass man zunächst einmal versucht hat, das Problem selbst zu lösen und zumindest in den Manpages gesucht hat. Um nicht immer die gleichen Fragen beantworten zu müssen, haben die Newsgroups normalerweise eine FAQ erstellt. FAQ bedeutet Frequently Asked Questions und ist eine Sammlung von Fragen und Antworten, die sich häufiger wiederholen. Es ist eine gute Idee, zunächst nach dieser FAQ zu fragen.

**Anfrage an
Newsgroups**

1.3.1 Unterteilung

Die Manpages sind in Abschnitte oder Sektionen unterteilt, die durchnummeriert sind.

1. Einfache Kommandos der Anwendungsebene
2. Systemaufrufe für Programmierer
3. Bibliotheksaufrufe für Programmierer
4. Spezialdateien für den Zugriff auf Hardware in `/dev`
5. Dateiformate und -aufbauten, beispielsweise `printcap`
6. Spiele
7. Makropakete und Konventionen. Hier findet sich beispielsweise eine Beschreibung von X oder die Definition der verschiedenen Zeichensätze.
8. Systemadministrationsbefehle (in der Regel nur für root)

Wenn ein Schlüsselwort in mehreren Sektionen auftaucht, wird die Seite angezeigt, die die niedrigere Sektionsnummer hat. Auf einigen Systemen kann man mit der Umgebungsvariablen `MANSECT` die Reihenfolge der Sektionen selbst festlegen. Dabei werden die Sektionen durch Doppelpunkte getrennt angegeben.⁷ Beispielsweise `3:2:1:4...`

⁷ vgl. Johnson, Troan: Anwendungen entwickeln unter Linux. Addison-Wesley, 1998. S. 75.

1.3.2 Sonderzeichen in Hilfetexten

Bei der Beschreibung der Befehle findet man in den Manpages Sonderzeichen, die die Art der Parametergestaltung beschreiben sollen. Auf den ersten Blick sehen sie recht verwirrend aus. Sie geben an, welche Parameter optional sind und zeigen Alternativen auf. Im Einzelnen gibt es:

Zeichen	Bedeutung
[optional]	»optional« kann, muss aber nicht angegeben werden.
{ dies das }	Hier muss entweder »dies« oder »das« stehen.
<variable>	Hier steht eine zu benennende Variable, also nicht das Wort variable

Tabelle 1.1 Metazeichen in man

In diesem Buch habe ich mit Absicht auf diese Syntax verzichtet. Dadurch wird die Beschreibung mancher Dinge zwar länger, aber hoffentlich auch verständlicher. Als Beispiel ist hier der Anfang der Manpage von man zu sehen.

man(1) Manual Hilfsprogramme man(1)

NAME

man - Programm zum Einsehen der Online-Manuale

SYNTAX

man [-acdhwtZV] [-m System[,...]] [-L locale] [-p
Zeichenkette] [-M Pfad] [-P Pager] [-r Prompt] [-T Format]
[-S Liste] [-e Erweiterung] [[Abschnitt] Seite ...] ...
man -l [-tZ] [-p Zeichenkette] [-P Pager] [-r Prompt] [-T
Format] Datei ...
man -k Schlüsselwort ...
man -f Seite ...

BESCHREIBUNG

man ist der Manualbrowser des Systems. Jedes Argument
Seite ist normalerweise der Name eines Programmes oder
einer Funktion. Gefunden und angezeigt wird die Manual
seite, die auf alle Argumente paßt. Wenn ein Abschnitt
angegeben wird, sucht man nur in diesem Abschnitt der Man

ualseiten. Ohne Angabe eine explizite Angabe werden alle verfügbaren Abschnitte in einer vorher definierten Reihenfolge durchsucht. Wenn die Seite in mehreren Abschnitten

lines 1-25

1.4 So sage ich es meinem UNIX

UNIX-Kommandos werden eingetippt und mit der Return-Taste abgeschlossen. Sie gehen an ein Programm, das man Shell nennt. Die Shell übernimmt die Interpretation der Befehle und ruft zur Ausführung das Betriebssystem oder die angesprochenen Programme auf.

Ein Befehl besteht zuerst aus dem Befehlsnamen. Dieser bezeichnet meist ein Programm. Es folgen ein oder mehrere Leerzeichen zur Abtrennung der Parameter. Auch die Parameter werden voneinander durch Leerzeichen getrennt. Parameter unterteilen sich in Optionen und Argumente.

Leerzeichen ist
Trennzeichen

Optionen sind an einem Minuszeichen zu erkennen. Sie bewirken eine Veränderung der Programmausführung. Werden mehrere Optionen mitgegeben, können diese direkt hintereinander geschrieben werden, und nur ein Minuszeichen muss am Anfang geschrieben werden. Statt `-l -a` kann man also auch `-la` schreiben. Neuere Programme verwenden gern Worte als Optionen. Zur Unterscheidung von kombinierten Optionen haben sie meistens zwei Bindestriche⁸. Schließlich haben die meisten Befehle Argumente. Dies sind die Objekte, auf denen die Befehle ausgeführt werden sollen, meist Dateien oder Verzeichnisse. Je nach Art des Befehls kann es gar keine oder beliebig viele Argumente geben.

Optionen

UNIX-Programme sind normalerweise schweigsam. Wenn das Programm keine besonderen Ausgaben hat und kein Fehler aufgetreten ist, erscheint oft keine Meldung, sondern nur der Prompt.⁹ Bei einem Fehler gibt es allerdings eine Fehlermeldung. Diese kommt vom aufgerufenen Programm, oder die Shell meldet sich, wenn sie den Befehl bzw. das Programm nicht kennt oder die Struktur des Befehls ihr nicht behagt.

Fehlermeldungen

So könnte nach Eingabe von `abcdefg` als Kommando eine Meldung wie die folgende erscheinen:

⁸ Eine Ausnahme bildet beispielsweise `find`, das Optionsworte, aber nur einen Bindestrich verwendet.

⁹ Als Prompt bezeichnet man das, was in Ruhestellung links neben Ihrer Eingabemarke steht.

```
sh: abcdefg: not found.
```

Da UNIX kein Programm namens `abcdefg` finden konnte, meldet der Kommandointerpreter `sh` den Fehler. `sh` heißt die Standard-Shell. Sie kann bei Ihnen vielleicht anders heißen. Kommt ein korrekt aufgerufenes Programm mit der Eingabe nicht zurecht, meldet es sich. Zum Beispiel erscheint nach der Eingabe `grep o p` die Meldung

```
sh: grep: 0652-033 cannot open p.
```

Vor der Fehlermeldung sieht man sehr schön die Aufrufhierarchie. `sh` rief `grep` auf, und `grep` meldet den Fehler. Das Programm `grep` gibt es also, und es wurde auch gestartet. `grep` hatte mit `o` keine Probleme, aber `p` wollte es wohl öffnen, konnte es aber nicht finden oder hatte kein Recht. die Datei zu öffnen.

Übrigens brauchen Sie nicht zu erschrecken. Die meisten UNIX-Umgebungen haben inzwischen auch deutsche Meldungen.

1.5 Operationen mit Dateien

Groß- und Kleinschreibung von Dateien

Eine Datei ist eine DATenEinheit, also ein Rudel Daten in einem Sack. Das können Texte, Programme oder Datenbanken sein. Eine Datei hat immer einen Namen. UNIX unterscheidet sehr genau zwischen Groß- und Kleinschreibung, auch in Dateinamen. Die Dateien **montag** und **Montag** können nebeneinander im gleichen Verzeichnis stehen und sind zwei voneinander völlig unabhängige Dateien. Mac OS X behandelt aus Kompatibilitätsgründen zum alten System Klein- und Großbuchstaben gleich. Dieser Effekt tritt ebenfalls auf, wenn man es mit an sich UNIX-fremden Systemen zu tun hat, die sich so verhalten. Dies betrifft beispielsweise SAMBA (siehe S. 390) oder Medien, die unter MS Windows erzeugt wurden.

Der Name einer UNIX-Datei hat im Gegensatz zu MS-DOS oder VMS oft keine durch einen Punkt abgetrennte Endung, die den Inhalt einer Datei beschreibt. Insbesondere haben ausführbare Dateien spezielle Endungen wie `.EXE`, `.COM` oder `.BAT`. Das kennt UNIX nicht. Ausführbare Dateien haben unter UNIX normalerweise keine Endung. Dagegen gibt es sie durchaus bei C-Programmquelltexten mit `.c`, Objekten mit `.o` oder HTML-Dokumenten mit `.html`.

Wer mit UNIX arbeitet, wird feststellen, dass dort fast alles kleingeschrieben wird. In Kombination mit der Kürze der Kommandos kann man schnell zu dem Ergebnis kommen, dass UNIX-Benutzer faul sind. Auf

jeden Fall ist es der UNIX-Anwender gewohnt, mit wenigen Tastenanschlägen maximale Leistung zu entfalten.

1.5.1 Eine kleine Beispielsitzung

Anhand einer kleinen Beispielsitzung sollen die ersten Befehle von UNIX erklärt werden. Am deutlichsten wird es, wenn Sie alles direkt ausprobieren. Dazu melden Sie sich zunächst einmal an. Als Erstes soll ein Verzeichnis angelegt werden, in dem man nach Herzenslust experimentieren kann.

```
mkdir spielwiese  
cd spielwiese
```

Der Befehl `mkdir` (`mkdir` steht für `make directory`) erzeugt das Verzeichnis **spielwiese**. Mit `cd` (`cd` steht für `change directory`) wechselt man hinein. Ein Verzeichnis ist wie ein Ordner, in dem man mehrere Dateien aufbewahrt. Bei manchen Systemen sehen Sie sogar den Namen **spielwiese** links neben Ihrem Cursor stehen. Als Erstes brauchen Sie eine Datei zum Spielen. Dazu kopieren Sie mit dem ersten Befehl eine Datei namens **passwd** aus dem Systemverzeichnis **/etc** hierher.

Verzeichnis
anlegen

```
cp /etc/passwd .  
cp passwd anton  
cp anton paula
```

Verzeichnisnamen werden durch einen normalen Schrägstrich (/ , der über der 7, auch *Slash* genannt) und nicht durch den umgekehrten Schrägstrich (\ , genannt *Backslash*) getrennt, wie das unter MS Windows oder MS-DOS üblich ist. Im ersten Befehl kopieren Sie aus dem Verzeichnis **/etc** die Datei **passwd** in das aktuelle Verzeichnis. Das aktuelle Verzeichnis wird mit einem Punkt bezeichnet. Jetzt steht im aktuellen Verzeichnis eine Datei **passwd**. Der zweite `cp`-Befehl kopiert die Datei **passwd** und gibt ihr den Namen **anton**. Da kein Schrägstrich im Argument vorkommt, bewegt sich alles im aktuellen Verzeichnis **spielwiese**. Zuletzt wird **anton** nach **paula** kopiert. Jetzt liegen drei Dateien im Verzeichnis. Das kann man sich anschauen, indem man den Befehl `ls` (`ls` steht für `list`) verwendet.

Der / ist der
Verzeichnistrenner

```
ls  
ls -l  
ls -l paula
```

Nach dem ersten Befehl haben Sie eine Liste von drei Namen nebeneinander (`passwd anton paula`). Das sind die Dateien, die Sie oben kopiert

Anzeigen der
Dateien

hatten. `ls` zeigt Dateinamen an. Kombiniert mit der Option `-l` zeigt `ls` wieder Dateinamen an, aber diesmal in der Langform. Unter UNIX ist es wie bereits erwähnt üblich, Befehle mit Optionen zu steuern. Diese Optionen bestehen aus einem Bindestrich und einem Buchstaben oder aus zwei Bindestrichen und einem Wort. Die Langform zeigt unter anderem das Datum der Entstehung und die Größe der Datei in Bytes. Im letzten Befehl wird nach der Option ein Dateiname angegeben. Der Befehl heißt: Zeige die Langform des Dateinamens der Datei **paula**.

Im nächsten Befehl soll ein wenig mit den Argumenten gespielt werden. Im ersten Befehl wird aufgelistet, welche beiden Dateien in Langform angezeigt werden sollen. Im zweiten Befehl wird ein Stern eingesetzt, der als Platzhalter für beliebig viele Zeichen verwendet wird.

```
ls -l passwd paula
ls -l p*
```

*** als Platzhalter** `p*` bedeutet: Zeige alle Dateien, die mit `p` anfangen. Das sind in diesem Fall **passwd** und **paula**. Der Stern bedeutet also: Setze an dieser Stelle beliebige Zeichen ein. Mit `p*d` sieht man die Datei **passwd**, da **passwd** mit `p` anfängt und mit `d` aufhört. `p*d*` findet ebenfalls die Datei **passwd**, da der Name mit `p` anfängt und ein `d` enthält, wenn auch an der letzten Stelle. Danach folgen beliebig viele Zeichen, in diesem Fall keine.

Die Shell verteilt die Sterne Wichtig zu erwähnen ist, dass nicht der Befehl `ls` den Stern interpretiert, sondern dies von der Kommandozeile, der Shell, getan wird. Dies ist deswegen wichtig, weil so garantiert ist, dass alle Befehle den Stern gleich interpretieren, weil sie ihn eben nicht selbst interpretieren, sondern die Shell dies tut. Sie liefert an das Programm eine Liste aller Dateien, die auf die Maske passen. Die Shell sucht beispielsweise bei `p*` alle Dateien, die mit `p` beginnen, und listet sie auf. Das Programm `ls` bekommt als Argument von der Shell **passwd paula** geliefert. Noch ein paar Befehle:

```
mv paula erna
rm anton
ls
```

Umbenennen und Löschen **paula** wurde nun in **erna** umbenannt. Der Befehl lautet `mv` für move (engl. bewegen). Man kann mit diesem Befehl also Dateien auch in andere Verzeichnisse schieben. **anton** ist nun verschwunden. `rm`, kurz für remove (engl. entfernen), hat diese Datei entsorgt. Zuletzt soll wieder aufgeräumt werden. Die folgenden Befehle löschen alle Dateien, wechseln wieder in das Verzeichnis zurück, aus dem Sie kamen, und entfernen das Übungsverzeichnis:

```
rm *  
cd ..  
rmdir spielwiese
```

Im ersten Befehl wird `rm *` von der Shell zu `rm erna passwd` expandiert. Der »waagerechte« Doppelpunkt (..) hinter dem `cd` bezeichnet das übergeordnete Verzeichnis. Man spricht auch vom Elternverzeichnis. Sie verlassen also Ihre **spielwiese** und stehen nun dort, wo Sie angefangen hatten. `rmdir` löscht leere Verzeichnisse. Da Sie mit dem `rm` vorher aufgeräumt haben, ist **spielwiese** leer und wird entsprechend auch gelöscht. Ein Verzeichnis kann nicht mit `rm` gelöscht werden¹⁰, sondern nur mit dem Befehl `rmdir`.

1.5.2 Dateien anzeigen: ls

Die Aufgabe von `ls` ist die Anzeige von Dateinamen. Es folgen Optionen, die mit einem Bindestrich beginnen, und dann die anzuzeigenden Dateien. Werden keine Angaben zu den Dateien gemacht, werden alle Dateien des aktuellen Verzeichnisses angezeigt.

Wird als Parameter der Name eines Verzeichnisses eingegeben, zeigt `ls` den kompletten Inhalt des angegebenen Verzeichnisses. Dies kann leicht irritierend sein, wenn man nur nähere Informationen zu einem Verzeichnis haben wollte. Mit der Option `-d` (d wie directory) kann man diesen Effekt unterbinden. Dann wird nur das Verzeichnis angezeigt und nicht hineingeschaut.

Verzeichnisse
anzeigen: `ls -d`

Beispielsweise legen Sie ein Verzeichnis **spielwiese** an und lassen es sich hinterher mit `ls spielwiese` anzeigen, um zu sehen, ob es noch da ist. Sie sehen – nichts. Der Grund liegt darin, dass `ls` das Verzeichnis **spielwiese** nicht selbst anzeigt, sondern seinen Inhalt. Da dieser bisher leer ist, wird die leere Liste angezeigt. Es sieht also so aus, als wäre **spielwiese** verschwunden. Mit `ls -ld spielwiese` kann man sie wiederum sichtbar machen.

Einige Optionen werden häufiger gebraucht. Die Langform mit `-l` zeigt alle Informationen zu den Dateien, die man sich wünschen kann. Von rechts nach links erkennt man den Dateinamen, es folgen der Zeitpunkt der Erstellung und die Größe der Datei. Das erste Zeichen der Zeile zeigt ein `d` für Verzeichnisse und ein Minuszeichen für eine normale Datei. Die

`ls -l` zeigt
Langform

¹⁰ Abgesehen von dem Spezialfall eines rekursiven Löschsens mit der Option `-r`, die später behandelt wird.

anderen Informationen werden in den späteren Kapiteln noch ausführlich erläutert. Hier sehen Sie ein Beispiel für die Ausgabe von `ls -l`.

```
-rwx----- 1 arnold users 13654 Jan 17 04:41 a.out
drwxr-x--- 2 arnold users 4096 Mär 6 23:35 awk
-rwxr-xr-x 1 arnold users 13856 Feb 8 10:48 copy
-rw-r--r-- 1 arnold users 175 Feb 8 10:48 copy.c
drwxr-xr-x 2 arnold users 4096 Jan 13 12:37 dir
-rw-rw-r-- 1 arnold users 98 Apr 13 13:55 doppel.c
-rwxr-xr-x 1 arnold users 13516 Feb 8 21:25 env
-rw-r--r-- 1 arnold users 165 Feb 8 21:25 env.c
drwx----- 2 arnold users 4096 Feb 21 14:13 ipc
drwxr-xr-x 2 arnold users 4096 Feb 13 13:51 make
-rwxr-xr-x 1 arnold users 13489 Apr 11 21:34 moin
-rw-r--r-- 1 arnold users 60 Apr 13 13:55 moin.c
```

Der Dateiname kann unter UNIX relativ frei gestaltet werden. Er darf keinen Schrägstrich enthalten, da dieser als Verzeichnistrenner interpretiert wird. Enthält der Dateiname Sonderzeichen, die die Shell interpretiert, muss man den Namen in Anführungszeichen setzen, wenn man ihn von der Kommandozeile aus benutzt. Alternativ kann man auch einen Backslash vor das Sonderzeichen stellen. Für die Shell bedeutet der Backslash, dass sie das folgende Zeichen nicht interpretieren, sondern einfach durchreichen soll.

Option	Wirkung
-l	zeigt alle Informationen über die Datei an
-a	zeigt auch die Dateien, die mit einem Punkt beginnen
-d	zeigt das Verzeichnis und nicht dessen Inhalt
-t	sortiert nach letzter Änderung. Neueste Dateien kommen zuerst
-r	dreht die Sortierreihenfolge um
-R	zeigt alle Unterverzeichnisse

Tabelle 1.2 Einige Optionen zu `ls`

ls -a zeigt alle Dateien

Die Option `-a` zeigt auch Dateien an, die mit einem Punkt beginnen. Unter UNIX werden Dateien, die zur Konfiguration verwendet werden, gern so benannt, dass das erste Zeichen ein Punkt ist. So stören sie nicht, wenn man eigentlich nur die Arbeitsdateien betrachten will. Auch ein versehentliches Löschen wird unwahrscheinlicher, weil der Befehl `rm *`

diese Dateien nicht erfasst. Will man `-l` und `-a` kombinieren, kann man `ls -a -l` oder einfacher `ls -al` angeben.

Mit der Option `-t` wird statt nach dem Alphabet nach der Zeit sortiert. Dabei erscheint zuerst die neueste Datei. Mit der Option `-r` wird die Reihenfolge umgekehrt.

ls -t sortiert nach der Zeit

Die Option `-R` zeigt eine Liste aller Unterverzeichnisse mit ihren Unterverzeichnissen.

ls hat erheblich mehr Optionen. Wer sie gern alle kennen lernen möchte, muss nur man `ls` eingeben.

1.5.3 Dateien kopieren: cp

Der Befehl `cp` kopiert Dateien an genau ein Ziel. Wird nur eine Datei kopiert, kann das Ziel ein Dateiname sein. Dann wird von der Quelldatei eine Kopie angefertigt und diese unter dem Zielnamen abgestellt. Werden dagegen mehrere Dateien kopiert, muss das Ziel ein Verzeichnis sein, da ja nicht alle Quelldateien den gleichen Namen bekommen können. Die Kopien finden sich nach der Ausführung unter ihrem bisherigen Namen im angegebenen Zielverzeichnis. Soll das Ziel das aktuelle Verzeichnis sein, muss der Punkt dafür angegeben werden.

cp kopiert Dateien

Um mehrere Dateien zu erfassen, kann man sie aufzählen oder durch einen Stern auswählen. Wichtig ist, dass der zuletzt angegebene Name von `cp` immer als Ziel interpretiert wird. Ein Verzeichnis als Quelle wird von `cp` einfach übergangen.

`cp` erzeugt beim Kopieren immer eine neue Datei. Darum hat eine Dateikopie auch immer den Zeitpunkt des Kopierens als Änderungsdatum, und nicht das Datum der Datei, von der sie kopiert wird. Auch der Eigentümer der neu entstandenen Datei ist immer der Anwender, der den Befehl aufgerufen hat. Dieses Verhalten kann mit der Option `-p` unterbunden werden, die aber nicht in allen UNIX-Versionen vorhanden ist.

cp erstellt neue Dateien

Auch die Möglichkeit, mit der Option `-r` komplette Verzeichnisbäume zu kopieren, ist nicht auf allen Systemen verfügbar. Darum wird das Kopieren kompletter Verzeichnisbäume unter Beibehaltung aller Eigenschaften auf älteren Systemen normalerweise mit dem Kommando `tar` realisiert, das an anderer Stelle betrachtet wird (siehe S. 88).

Welche Möglichkeiten `cp` auf Ihrem System sonst noch bietet, erfahren Sie wieder mit man `cp`.

1.5.4 Dateien verschieben oder umbenennen: mv

mv verschiebt und benennt um

Mit dem Befehl `mv` kann man einer Datei einen anderen Namen geben oder mehrere Dateien in ein anderes Verzeichnis schieben. In den meisten Fällen fasst `mv` die Dateien selbst gar nicht an, sondern verändert nur die Einträge in den Verzeichnissen. Darum wird das Verschieben selbst großer Dateien erstaunlich schnell erledigt. Eine Ausnahme gibt es nur, wenn Ursprung und Ziel auf verschiedenen Dateisystemen liegen, beispielsweise auf verschiedenen Festplatten. Dann wird zunächst kopiert, dann wird das Original gelöscht, und zu guter Letzt werden die Eigenschaften der Originale übernommen.

1.5.5 Dateien löschen: rm

rm -i löscht nur nach Rückfrage

Der Befehl `rm` löscht Dateien. Dies geschieht unwiderruflich, und darum ist es ganz gut, dass es die Option `-i` gibt. Dann fragt `rm` bei jeder einzelnen Datei nach, ob sie wirklich entfernt werden soll.

Will man einen kompletten Verzeichnisbaum löschen, verwendet man die Option `-r`. Damit geht `rm` rekursiv den gesamten Baum durch und entfernt alle enthaltenen Dateien und Verzeichnisse.

1.5.6 Verzeichnisbefehle: mkdir, rmdir, cd und pwd

Um Dateien zu ordnen und zu gruppieren, legt man Verzeichnisse an. Der Befehl zum Erzeugen von Verzeichnissen lautet `mkdir`.¹¹

cd wechselt das Verzeichnis

Der Anwender befindet sich immer in einem Verzeichnis. Mit `cd` wechselt er das Verzeichnis. In diesem arbeitet er ab sofort, darum spricht man auch von seinem Arbeitsverzeichnis. `cd` ohne Parameter wechselt in das Heimatverzeichnis. Das Heimatverzeichnis ist das Verzeichnis, in dem sich der Anwender nach dem Anmelden zunächst befindet.

pwd zeigt den aktuellen Pfad an

Um zu ermitteln, in welchem Verzeichnis man sich aktuell befindet, verwendet man den Befehl `pwd` (print work directory).

rmdir löscht leere Verzeichnisse

Um ein Verzeichnis wieder verschwinden zu lassen, verwendet man `rmdir` (remove directory). Allerdings kann man mit `rmdir` nur leere Verzeichnisse löschen und auch nur solche, die derzeit nicht verwendet werden. In Verwendung sind auch Verzeichnisse, in denen noch eine Sitzung von einem anderen Terminal stattfindet. Man darf nämlich nicht einfach jemand anderem das Verzeichnis unter den Füßen wegziehen. Leider ist es nicht

¹¹ Die Abkürzung `md` für `mkdir` gibt es unter UNIX nicht. Wer sie vermisst, kann sie sich aber als alias (siehe S. 99) definieren.

immer ganz leicht herauszufinden, wer mit welchem Prozess dort gerade arbeitet. Hier helfen die Befehle `fuser` und `ps` (siehe S. 253) und auch `ls` (siehe S. 254).

1.6 Der UNIX-Verzeichnisbaum

Eine UNIX-Maschine verteilt ihre Dateien in einen Verzeichnisbaum. Dieser erstreckt sich oft über mehrere physikalisch unterschiedliche Laufwerke. Es gibt also keine Laufwerksbuchstaben oder benannte Laufwerke. Will man beispielsweise eine Diskette bearbeiten oder lesen, muss sie in den Verzeichnisbaum eingehängt werden (siehe S. 174).

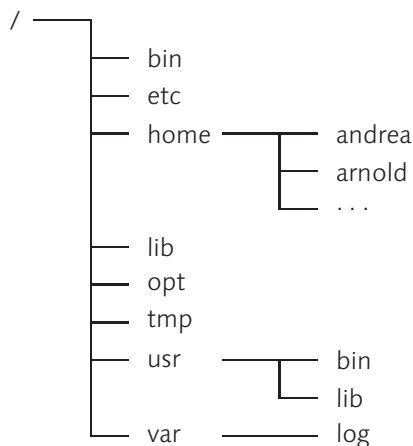


Abbildung 1.1 Verzeichnisbaum

1.6.1 Navigation

Ein Anwender befindet sich immer an einer Position im Verzeichnisbaum. Dieses Arbeitsverzeichnis zeigt der Befehl `pwd`. Der Befehl `cd` wechselt die Position im Verzeichnisbaum.

Der Schrägstrich ist der Verzeichnistrenner, aber auch die Bezeichnung für den Ursprung, die Wurzel aller Verzeichnisse. Mit `cd /` wechselt man in das Wurzelverzeichnis des Verzeichnisbaums.

**/ ist die Wurzel
des Baumes**

Jede Datei eines UNIX-Systems ist vom Wurzelverzeichnis aus zu erreichen. Eine vollständige Dateibezeichnung beschreibt dessen Position von der Wurzel aus. Beispielsweise hieß die erste Version des Textes dieses Buches inklusive Pfad:

`/home/arnold/my/texte/unixbuch.sdw`

Befindet man sich allerdings im Verzeichnis **/home/arnold**, dann kann man dieselbe Position auch kürzer bezeichnen, nämlich mit:

my/texte/unixbuch.sdw

**Absolute
Pfadnamen
beginnen mit /**

Fehlt also am Anfang eines Verzeichnispfades der Schrägstrich, ist die Pfadbezeichnung relativ zum aktuellen Arbeitsverzeichnis. Steht dagegen ein / am Anfang, ist die Bezeichnung absolut, egal in welchem Verzeichnis man sich soeben befindet.

Der Punkt bezeichnet das aktuelle Arbeitsverzeichnis, in dem man sich derzeit aufhält. Zwei Punkte hintereinander bezeichnen das im Verzeichnisbaum oberhalb liegende Verzeichnis. Ist das aktuelle Verzeichnis wie oben **/home/arnold**, bezeichnet **..** das Verzeichnis **/home**.

**~ bezeichnet das
Heimatverzeichnis**

Ein weiteres wichtiges Verzeichnis ist das Heimatverzeichnis. Das ist das Verzeichnis, in dem man sich direkt nach dem Einloggen befindet. Gibt man **cd** ohne Parameter an, wechselt man dort hin. Will man das Heimatverzeichnis in einem Befehl angeben, verwendet man die Tilde (**~**).¹² Ist also **/home/arnold** das Heimatverzeichnis, kann man die Beispieldatei von oben auch so bezeichnen:

~/my/texte/unixbuch.sdw

1.6.2 Ein Blick unter die Haube: i-nodes

**i-nodes halten die
Informationen
über eine Datei**

In der UNIX-Literatur und auch in den Manpages wird häufiger der Begriff i-node genannt. Der i-node ist ein Beschreibungsblock in der Dateisystemstruktur für eine Datei. Dort stehen sämtliche Informationen, die man auch bei einem **ls -l** sehen kann, und noch ein paar Verwaltungsinformationen mehr. Im i-node befindet sich ein Verweis auf die tatsächlichen Datenblöcke der Datei. Die i-nodes sind durchnummeriert, und diese Nummer ist auf dem Dateisystem eindeutig.

**Im Verzeichnis
steht nur die
i-node Nummer**

Da nun alle Informationen über die Datei im i-node stehen, braucht man sie nicht in das Verzeichnis zu schreiben. Tatsächlich enthält der Verzeichniseintrag einer Datei auch nur wenige Informationen, in erster Linie den Dateinamen und die Nummer des i-nodes, in dem dann alle weiteren Informationen über die Datei stehen.

¹² Streng genommen ist die Umsetzung der Tilde auf das Heimatverzeichnis ein Service der Shell.

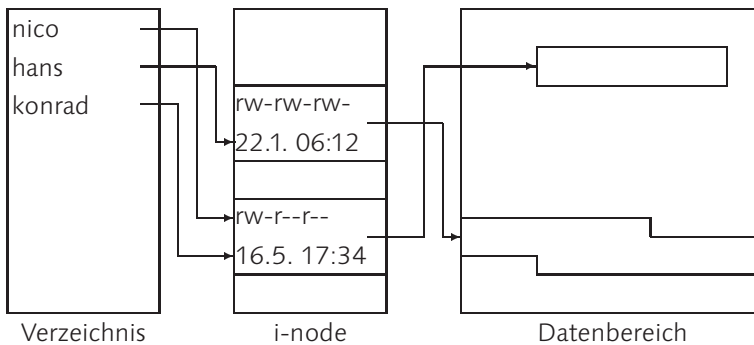


Abbildung 1.2 Verzeichnis, i-node und Datenbereich

Diese Vorgehensweise hat den Vorteil, dass Verzeichniseinträge relativ klein sind und im Verhältnis zu anderen Systemen erstaunlich flott verwaltet werden können. Auch die Möglichkeit, mit Links zu arbeiten (siehe S. 52), also dass zwei verschiedene Verzeichniseinträge auf dieselbe Datei zeigen, ist mit Hilfe des i-nodes leicht implementierbar. Aus dieser Architektur wird aber auch klar, dass es zwar mehrere Namen für dieselbe Datei gibt, aber dennoch nur einen Besitzer, da alle Verweise auf denselben i-node zeigen.

Vorteile

1.6.3 Was ist wo?

Es gibt bestimmte Verzeichnisse, die man auf jeder UNIX-Maschine findet.

/etc

Dies war auf den älteren Systemen das Arbeitsverzeichnis des Systemadministrators. Hier lagen alle Konfigurationsdateien des Systems und die Programme, die nur root verwenden durfte. Im Laufe der Zeit wurde es in **/etc** etwas eng, sodass die Administrationswerkzeuge nach **/sbin** ausgelagert wurden.

Konfiguration

In **/etc** befinden sich immer noch die Konfigurationsdateien und die Startskripten von UNIX.

/bin und /lib

Im Verzeichnis **/bin** befinden sich die Programme, die als minimale Ausstattung erforderlich sind, um das System zu administrieren. Dieses Verzeichnis sollte auf der Bootpartition liegen. Ebenfalls dort wird das Verzeichnis **/lib** als Ort für die dynamischen Bibliotheken der Programme aus **/bin** benötigt. Dynamische Bibliotheken sind Dateien mit Programmtei-

Basisbefehle und Bibliotheken

len, die von mehreren Programmen benötigt werden und so vom System zur Verfügung gestellt werden, sobald sie gebraucht werden.

/tmp

Temporäre Dateien

Das Verzeichnis **/tmp** ist für temporäre Dateien gedacht. Es ist für jeden Benutzer schreib- und lesbar und immer an dieser Stelle zu finden. Man sollte dieses Verzeichnis tatsächlich nur für temporäre Dateien verwenden, da einige Maschinen in regelmäßigen Abständen dieses Verzeichnis per cron (siehe S. 111) löschen.

/usr

Standard- installationen

Hier befinden sich die normalen Anwenderprogramme (**/usr/bin**) und deren Bibliotheken (**/usr/lib**). In **/usr/include** befinden sich die Headerdateien für die Schnittstellen zum System. Headerdateien werden von Programmierern benötigt, um Zugriffe auf die Systemaufrufe zu bekommen. Unter **/usr/man** sind die Manpages abgelegt. **/usr/X11** enthält die Programme zur grafischen Oberfläche.

Auf älteren Systemen sind hier auch die Verzeichnisse zu finden, die heute unter **/var** liegen. Bei einigen Systemen findet man aus Kompatibilitätsgründen noch symbolische Links auf Verzeichnisse in **/var**. Diese Umschichtung verfolgt das Ziel, dass im Verzeichnis **/usr** nicht geschrieben und gelöscht werden soll. Der Hauptgrund liegt darin, dass auf das Verzeichnis **/usr** oft zugegriffen wird. Wenn es durch Schreiben und Löschen fragmentiert wird, leidet der Durchsatz der Maschine. In der ständig aktuellen Diskussion über Viren könnte man auch überlegen, das Verzeichnis komplett schreibzuschützen. Aber so einfach ist das auch nicht. Zunächst müsste für Installationen von Programmen immer wieder der Schreibschutz aufgehoben werden. Des Weiteren hat auch jetzt schon ein normaler Anwender keinerlei schreibende Zugriffsrechte auf das Verzeichnis. Ist aber erst einmal der Zugang des root geknackt, gibt es ohnehin keinen Schutz mehr für die Maschine.

/var

Protokolle und Spooling

Hier liegen die Dateien, die vom System oder den Anwenderprogrammen verändert werden: Protokolle und Spooling. In älteren Systemen finden sich diese Dateien noch im Verzeichnis **/usr**.

Das Verzeichnis **/var/log** enthält Fehlerprotokolle. Die wichtigste Datei ist die **/var/log/messages**. Üblicherweise legt der Fehlerdämon syslogd

hier seine Meldungen ab. Allerdings ist der Ort dieser Datei durch die **/etc/syslog.conf** konfigurierbar (siehe S. 239).

Unter **/var/spool** finden sich die Verzeichnisse für das Spooling. Druckaufträge, Mails und Arbeitsaufträge (cron bzw. at, siehe S. 111) werden hier abgelegt.

/home bzw. /users

An das Verzeichnis **/home** werden bei den meisten UNIX-Systemen die Verzeichnisse der Benutzer angehängt. Unter Mac OS X heißt dieses Verzeichnis **/users**. Für jeden Benutzer wird ein eigenes Verzeichnis angelegt, und der normale Anwender hat mit seinen Dateien in den anderen Verzeichnissen normalerweise nichts verloren. Anfangs hatte man die Benutzerverzeichnisse unter **/usr** abgelegt. Als dieses Verzeichnis überquoll, verwandte man kurzzeitig **/user**. Da beide Verzeichnisse sprachlich schwer unterscheidbar waren, ging man schließlich zum Namen **/home** über. Festgelegt wird der Ort von Benutzerverzeichnissen durch die Datei **/etc/passwd** (siehe S. 144).

Heimatverzeichnis
der Benutzer

/opt

Inzwischen werden auch größere Programmpakete nicht mehr unter **/usr**, sondern unter **/opt** abgelegt.

Programmpakete

1.6.4 Suche im Verzeichnisbaum: find

Mit dem Kommando **find** ist es möglich, einen Verzeichnisbaum rekursiv¹³ nach Dateien zu durchsuchen und diese auszugeben oder Aktionen auf den gefundenen Dateien auszuführen. **find** ist also ein sehr mächtiger Befehl. Soll nur eine Datei gesucht werden, reicht die vereinfachte Syntax:

Suche einer Datei

```
find Pfadname -name Dateimaske -print
```

Der Pfadname gibt an, ab welcher Stelle die rekursive Suche beginnt. Die Dateimaske beschreibt die gesuchte Datei. Es können die normalen Wildcards verwendet werden, um mehrere Dateien anzusprechen.

¹³ Rekursiv heißt »selbstaufrufend« und ist eine besondere Programmier-technik. Da das Durchlaufen von Bäumen durch rekursive Programme am einfachsten zu realisieren ist, nennt man Bäume »rekursive Strukturen« und das Durchlaufen der kompletten Äste eines Baumes »rekursiv«. Vgl. Wirth, Niklaus: Algorithmen und Datenstrukturen mit Modula-2. 4. Aufl. Stuttgart, Teubner, 1986. S. 174ff.

Kommandos auf gesuchte Dateien

Neben der flexiblen Suche ist die interessanteste Fähigkeit von `find` die Ausführung eines Kommandos auf jeder der gefundenen Dateien. Die zuständige Option heißt `-exec`. Dahinter wird ein Befehl gesetzt, der mit einem Semikolon abgeschlossen wird. Letzteres muss durch einen Backslash gegen die Interpretation durch die Shell geschützt werden. Die Shell würde das Semikolon als Trennzeichen zwischen zwei Befehlen interpretieren, die nacheinander ausgeführt werden. Vergißt man den Backslash, würde die Shell das Semikolon selbst interpretieren und nicht an `find` weiterreichen. Der Bezug auf die gefundene Datei wird im Befehl durch ein Paar geschweiffter Klammern repräsentiert. Um beispielsweise alle Dateien namens **core** in den Heimatverzeichnissen zu löschen, lautet der Befehl:

```
find /home -name core -exec rm {} \;
```

Der erste Parameter gibt an, wo die Suche beginnen soll. `-name` fordert, dass ein Dateiname, der unter dem Verzeichnis **/home** gefunden wird, der Dateimaske entspricht. Nur dann wird die Datei von `find` weiter bearbeitet. Man kann es auch so ausdrücken, dass die Optionen einen Wahrheitswert zurückliefern müssen, der wahr ist, wenn die weiteren Optionen bearbeitet werden sollen. Die Maske zu `-name` ist einfach **core**. Nur Dateien mit dem Namen **core** werden also weiter bearbeitet. Zwischen der Option `-exec` und dem `\;` steht der Befehl, der auf die gefundenen Dateien angewandt wird. Der gefundene Dateiname mit seinem Pfad wird dort eingesetzt, wo sich im Befehl das geschweifte Klammernpaar befindet. Wenn `find` eine Datei namens **core** im Verzeichnis **/home/arnold** fände, würde der folgende Befehl generiert und per `-exec` ausgeführt:

```
rm /home/arnold/core
```

Der Befehl `find` kennt noch einige Optionen, die teils die Suche nach speziellen Dateien ermöglichen und teils die Ausgabe informativer gestalten. Wie oben schon angedeutet, hängt es vom dem Wahrheitswert, den die Optionen zurückliefert ab, ob die gefundene Datei weiterbearbeitet wird. Hier sind die wichtigsten Optionen zusammengefasst:

- ▶ **-name *Dateimaske*** ist wahr, wenn die Dateimaske mit dem gefundenen Dateinamen übereinstimmt. Es können reguläre Ausdrücke verwendet werden.
- ▶ **-print** ist immer wahr. Bewirkt, dass das Ergebnis der Suche angezeigt wird. Bei einigen `find`-Versionen muss `-print` nicht explizit angegeben werden, wenn keine weitere Option verwendet wird.

- **-ls** ist immer wahr. Es bewirkt, dass neben dem Pfadnamen noch ein wenig Statistik angezeigt wird. Dies beinhaltet die folgenden Angaben:
 - i-node Nummer der Datei (siehe S. 40)
 - Größe in Kilobytes (1024 Bytes)
 - Protection mode
 - Anzahl der »harten« Links
 - Benutzer
 - Gruppe
 - Größe in Bytes
 - Änderungszeitpunkt
- **-type *Zeichen*** Der Dateityp wird erfragt. Das Zeichen kann folgende Werte haben:

Zeichen	Bedeutung
f	Dateien
d	Verzeichnisse
l	symbolische Links
b c	Peripheriedateien
p s	Kommunikationsendpunkte (named pipe und socket)

Tabelle 1.3 Dateitypen

- **-perm *Oktalzahl*** Erfragt die Berechtigung, die in oktaler Darstellung angegeben wird. Zur Codierung siehe das Kapitel »Berechtigungen« auf S. 48. Beispiel:


```
find . -perm 0600 -print
```

Es werden alle Dateien unterhalb des aktuellen Verzeichnisses angezeigt, die Schreib- und Leserecht für den Benutzer und keine Zugriffsrechte für alle anderen zulassen.

Man kann festlegen, dass mindestens die angegebenen Rechte vorhanden sein müssen. Dazu muss der Zahl ein Minuszeichen vorangehen.
- **-exec *Befehl*** Führt einen Befehl aus. Die aktuelle Datei wird durch das geschweifte Klammernpaar { } als Argument gekennzeichnet. Der Befehl hört mit einem Semikolon auf. Damit die Shell es nicht selbst interpretiert, muss es mit einem Backslash versehen werden. Beispiel:

```
-exec rm { } \ ;
```

Logische Verknüpfung der Optionen

Man kann die verschiedenen Optionen von `find` kombinieren. Man könnte beispielsweise nach Dateien suchen, die **.rhosts** heißen und nicht die Berechtigung 600 haben. Dazu muss man die Aussagen miteinander logisch verknüpfen. Nach den Gesetzen der Logik heißt UND, dass beide Aussagen zutreffen müssen, und ODER, dass mindestens eine der Aussagen zutreffen muss. Derartige Konstruktionen werden vor allem im Bereich der Shellprogrammierung benötigt. Mit Hilfe der folgenden Operatoren sind logische Ausdrücke kombinierbar:

► ()

Eine Gruppe von logischen Ausdrücken und Operatoren in Klammern. Klammern sind besondere Zeichen für die Shell. Darum muss ihnen ein Backslash vorangestellt werden.

► !

Die Negation eines logischen Ausdrucks. Das Ausrufezeichen (!) repräsentiert den NICHT-Operator.

► -a

Die UND-Verknüpfung von logischen Ausdrücken. Werden mehrere logische Ausdrücke hintereinander geschrieben, wird dies implizit als UND interpretiert. Man kann auch explizit den Operator `-a` verwenden.

► -o

Die ODER-Verknüpfung von logischen Ausdrücken wird durch den Operator `-o` erreicht.

Hier sehen Sie zunächst den Befehl, der das obige Beispiel aufgreift und alle Dateien namens **.rhosts** in den Heimatverzeichnissen löscht, die nicht die Berechtigung 600 haben:

```
find /home -name .rhosts -a ! -perm 0600 -exec rm { } \ ;
```

Das folgende Beispiel sucht ab dem aktuellen Verzeichnis alle Dateien, die auf `.log` oder `.aux` enden:

```
find . -name \*.log -o -name \*.aux
```

Vor den Sternen muss ein Backslash stehen, damit sie an `find` durchgeht und nicht bereits durch die Shell interpretiert werden.

1.7 Dateieigenschaften

Der Begriff Datei ist unter UNIX sehr allgemein. Eigentlich ist fast alles als Datei ansprechbar – seien es Gerätedateien (special files), die den

Kontakt zur Peripherie herstellen, oder Verzeichnisse, die auch nur eine Sonderform von Dateien sind. Alle haben jedenfalls Eigentümer, Rechte und andere Eigenschaften.

1.7.1 Eigentümer wechseln: **chown** und **chgrp**

Jede Datei hat in ihrem Dateieintrag die Information, welchem Benutzer und welcher Gruppe sie gehört. Um genau zu sein, werden die User-ID und die Group-ID des Benutzers abgelegt, wie sie in der `/etc/passwd` des lokalen Rechners hinterlegt sind. Damit in einem lokalen Netzwerk die Eigentümerbezeichnungen konsistent sind, ist es wichtig, dass Benutzer auf jeder Maschine, auf der sie angemeldet sind, die gleiche User-ID haben. Gleiches gilt für die Group-ID.

**Jede Datei kennt
ihr Herrchen**

Eigentümer und Gruppe werden in den Eigenschaften einer Datei gespeichert, weil man für den Eigentümer und die Gruppe einer Datei jeweils andere Berechtigungen einstellen kann als für den Rest der Welt.

Eine neue Datei erhält immer die User-ID und die Standard-Group-ID des Anwenders, der sie erzeugt. Dies gilt beispielsweise auch, wenn eine Datei kopiert wird. Aus Sicht des Systems wird beim Kopieren eine neue Datei angelegt, die den Inhalt der alten Datei hat. Bei `mv` liegt der Fall anders, weil hier nur der Verzeichniseintrag an eine andere Stelle bewegt wird.

Um einer Datei oder einem Verzeichnis einen neuen Besitzer zuzuordnen, gibt es den Befehl `chown` (change owner). Der erste Parameter ist der neue Benutzername. Es folgen die Dateien, die dem Benutzer zugeordnet werden sollen. Zur Änderung des Eigentümers sind nur der bisherige Eigentümer und der Systemadministrator `root` berechtigt.

chown

```
chown Benutzer Dateien
```

Analoges gilt für `chgrp` (change group). Eine Datei gehört nicht nur einem Benutzer, sondern auch einer Gruppe. Auf diese Weise kann einer beschränkten Gruppe von Personen ein Recht auf die Datei zugewiesen werden, das die anderen Benutzer nicht haben. Die Group-ID der Datei kann mit dem Befehl `chgrp` geändert werden. Berechtigt ist ein Mitglied der Gruppe oder `root`.

chgrp

chgrp Gruppe Dateien

Man kann auch Benutzer und Gruppe bei einigen Varianten von chmod mit einem Befehl setzen.

chown Benutzer:Gruppe Dateien

1.7.2 Berechtigungen: chmod

chmod Jede Datei hat Informationen darüber, ob sie gelesen, geschrieben oder ausgeführt werden kann. Diese Rechte existieren jeweils für den Eigentümer, die Gruppe und alle restlichen Benutzer. Durch den Befehl `ls -l` kann man die Rechte von Dateien als eine Gruppe von neun Buchstaben oder Minuszeichen sehen.

Besitzer	Gruppe	Rest
r w x	r w x	r w x

Die Buchstaben bedeuten bei Dateien:

Recht	Bedeutung für Dateien
x	Die Datei ist ausführbar, also ein Programm oder ein Skript.
r	Die Datei darf gelesen werden.
w	Die Datei darf geschrieben werden.

Tabelle 1.4 Die Rechte für Dateien

Die Ausgabe eines `ls -l` zeigt die Rechte ab dem zweiten Zeichen in der oben genannten Reihenfolge für den Benutzer, die Gruppe und den Rest der Welt.

```
-rwx----- 1 arnold users 13654 Jan 17 04:41 a.out
drwxr-x--- 2 arnold users 4096 Mär 6 23:35 awk
-rwxr-xr-x 1 arnold users 13856 Feb 8 10:48 copy
-rw-r--r-- 1 arnold users 175 Feb 8 10:48 copy.c
drwxr-xr-x 2 arnold users 4096 Jan 13 12:37 dir
-rw-rw-r-- 1 arnold users 98 Apr 13 13:55 doppel.c
-rwxr-xr-x 1 arnold users 13516 Feb 8 21:25 env
-rw-r--r-- 1 arnold users 165 Feb 8 21:25 env.c
drwx----- 2 arnold users 4096 Feb 21 14:13 ipc
```



```
drwxr-xr-x    2 arnold  users      4096 Feb 13 13:51 make
-rwxr-xr-x    1 arnold  users     13489 Apr 11 21:34 moin
-rw-r--r--    1 arnold  users         60 Apr 13 13:55 moin.c
```

Ein paar Dateien sollen herausgegriffen werden. Die Datei **a.out** erlaubt dem Besitzer, aber sonst niemandem, sie zu schreiben, zu lesen und auszuführen. Dagegen darf die Datei **copy** zusätzlich sowohl von der Gruppe users als auch vom Rest der Welt gelesen und gestartet, aber nicht beschrieben werden. Die Datei **copy.c** dagegen ist ein Quelltext mit ganz typischen Rechten für Datendateien. Der Besitzer darf sie lesen und schreiben, alle anderen dürfen sie nur lesen.

Die Rechte werden als Oktalzahlen¹⁴ dargestellt. Das heißt, dass jeweils eine rwx-Gruppe auf eine Ziffer abgebildet wird. So ergibt sich:

r	w	x	Kennzahl	Bedeutung
0	0	0	0	keine Rechte
0	0	1	1	Ausführen, aber weder lesen noch schreiben
0	1	0	2	Nur Schreibrecht
0	1	1	3	Schreib- und Ausführungsrecht, nicht lesen
1	0	0	4	Nur Leserecht
1	0	1	5	Lese- und Ausführungsrecht
1	1	0	6	Lese- und Schreibrecht, kein Ausführungsrecht
1	1	1	7	Lese-, Schreib- und Ausführungsrecht

Tabelle 1.5 Oktalkodierung der Dateizugriffsrechte

Eine Besonderheit ergibt sich für Verzeichnisse. Hier bedeutet das *Fehlen(!)* einer Berechtigung:

Recht	Bedeutung für Verzeichnisse
x	In dieses Verzeichnis darf nicht gewechselt werden.
r	Die Dateinamen des Verzeichnisses können nicht gelesen werden.
w	Es darf keine Datei gelöscht, umbenannt oder hinzugefügt werden.

Tabelle 1.6 Die Rechte für Verzeichnisse

¹⁴ Oktalzahlen sind eine Sonderform der dualen Zahlendarstellung, in der 3 Bits eine Ziffer ergeben. Mit 3 Bits lassen sich die Zahlen von 0 bis 7 darstellen.

Auf den ersten Blick erscheinen die Unterschiede zwischen den Datei- und den Verzeichnisrechten kompliziert. Wenn man sich ein Verzeichnis aber als eine Datei vorstellt, in der die Dateinamen abgelegt sind, ist der Zusammenhang mit den Schreib- und Leserechten logisch. Wenn man das Verzeichnis nicht lesen kann, kann man auch die Dateinamen nicht anzeigen. Darf das Verzeichnis nicht geschrieben werden, kann man auch keine Einträge hinzufügen oder löschen oder umbenennen.

Die Rechte von Dateien oder Verzeichnissen können mit dem Kommando `chmod` (change mode) verändert werden. Beispiel:

```
chmod 754 meinskript
```

Mit diesem Befehl erhält die Datei **meinskript** die Rechte `rxwx-r-xr--`. Das bedeutet, dass der Eigentümer die Datei lesen, schreiben und ausführen darf. Die Mitglieder der besitzenden Gruppe dürfen die Datei einsehen und starten, der Rest der Welt darf die Datei nur lesen.

Auftritt unter fremder Lizenz: User-ID-Bit

Ausführen unter
der ID des
Besitzers

Bei Programmen kann beim `chmod` vor die drei Ziffern eine 4 gestellt werden. Das bewirkt, dass das Programm¹⁵ unter der Benutzerkennung des Eigentümers ausgeführt wird, egal, wer das Programm gestartet hat. Man spricht vom Set-User-ID-Bit oder vom SUID. Dieses wird verwendet, wenn auf bestimmte Daten nur mit dem entsprechenden Programm zugegriffen werden soll. Die Zugriffsrechte der Daten können auf 600 gesetzt werden, sodass sie nur der Eigner verändern darf. Ein Programm, das das User-ID-Bit gesetzt hat, darf niemals für jemand Fremdes beschreibbar sein.¹⁶ Beispiel für das Setzen des User-ID-Bits:

```
chmod 4755 myprog
```

Die Datei **myprog** würde durch den Befehl `ls -l` folgendermaßen angezeigt werden:

```
-rwsr-xr-x  1 arnold  users          0 Dez  7 00:10 myprog
```

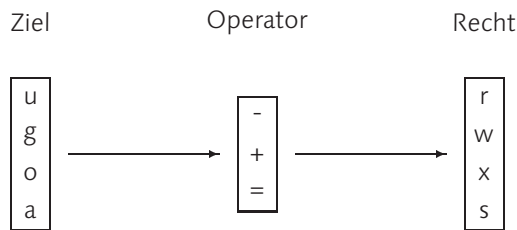
Statt dem `x` steht also ein `s` bei den Benutzerrechten.

¹⁵ Das funktioniert nicht bei Skripten, da diese von dem jeweiligen Interpreter »gelesen« werden und nicht selbst zur Ausführung kommen.

¹⁶ Herauszufinden, warum das so ist, ist eine Übungsaufgabe für den interessierten Leser.

Buchstaben statt Zahlen

Wem das Rechnen in Dualzahlen gar nicht liegt, der hat auch die Möglichkeit, chmod mit Buchstaben zu steuern. In Form einer Zuweisung wird auf der linken Seite des Operators die Zielgruppe genannt. Auf der rechten Seite stehen die Rechte. Als Zielgruppe gibt es den Eigentümer (u für user), die Gruppe (g) und alle übrigen (o für others). Mit einem a (a für alle) oder durch Weglassen der Zielgruppe kann man alle Zielgruppen auf einmal ansprechen.



Das Verändern eines Rechtes wird durch einen Ausdruck herbeigeführt, der sich aus den in der Grafik gezeigten drei Balken zusammensetzt. Das Gleichheitszeichen bewirkt ein Setzen, das Minuszeichen ein Entziehen und das Pluszeichen das Hinzufügen eines Rechtes. Dass die Rechte mit r, w, und x vergeben werden, ist wenig überraschend. Das s ist das User-ID-Bit. Beispiele für die Ausdrücke, die statt der Oktalzahl beim chmod verwendet werden können, finden Sie in der folgenden Tabelle:

Term	Bedeutung
a=r	Alle Anwender dürfen nur lesen.
u+w	Der Besitzer erhält zusätzlich Schreibrecht.
o-r	Der »Welt« wird das Leserecht entzogen.
go=rx	Die Gruppe und die »Welt« erhalten das Lese- und das Ausführungsrecht.

Tabelle 1.7 Beispiele für chmod-Optionen

Der Nachteil der Buchstabenschreibweise ist, dass manche Kombination mit zwei Befehlen gegeben werden muss. Soll eine Datei so gesetzt werden, dass der Besitzer lesen und schreiben, alle anderen aber nur lesen sollen, dann ist 644 natürlich kürzer, als zuerst a=r und im zweiten Schritt u+w zu schreiben. Dafür ist es mit Ziffern recht umständlich, alle Ausführungsrechte der Dateien im aktuellen Verzeichnis zu entziehen und dabei die anderen Rechte nicht zu verändern.

Maske für die Rechte neuer Dateien: umask

umask filtert
Rechte.

Mit dem Befehl `umask` wird festgelegt, welche Rechte beim Anlegen von Dateien nicht gesetzt werden dürfen. Normalerweise gibt ein Programm beim Anlegen einer Datei an, welche Rechte die Datei haben soll. Diese Anforderung wird durch den Wert von `umask` gefiltert. Im Allgemeinen ist dieser Wert `022` und braucht nur in den seltensten Fällen geändert zu werden. Der Wert `022` bedeutet, dass beim Erstellen von Dateien für die Gruppe und für den Rest kein Schreibrecht angelegt wird. Der Befehl `umask` ohne Parameter zeigt den geltenden Wert. Wird ein Parameter angegeben, wird die Maske entsprechend geändert.

```
umask 002
```

Mit diesem Befehl wird verhindert, dass neu erzeugte Dateien vom Rest der Benutzer verändert werden. Der Besitzer und die Gruppe erhalten dadurch das Schreibrecht.

1.7.3 Neuer Zeitstempel: touch

Der Befehl `touch` (engl. berühren) aktualisiert die Zugriffszeit einer Datei auf den jetzigen Zeitpunkt. Wenn die Datei bereits existierte, ist der Effekt derselbe, als wäre sie neu, aber unverändert geschrieben worden.

```
gaston> ls -l moin.c
-rw-r--r--  1 arnold  users           60 Apr 13 13:55 moin.c
gaston> touch moin.c
gaston> ls -l moin.c
-rw-r--r--  1 arnold  users           60 Jun 29 10:13 moin.c
gaston> date
Sam Jun 29 10:13:17 CEST 2002
gaston>
```

Dateien anlegen

Gab es die angegebene Datei bis jetzt noch nicht, wird sie angelegt. Sie ist dann leer, hat also eine Größe von 0 Byte. Die Möglichkeit, Dateien anzulegen, ist beispielsweise für Protokolldateien wichtig. Die meisten Programme, die in solche Dateien schreiben, sind nicht in der Lage, die Datei auch zu erzeugen.

1.7.4 Links: Zwei Namen, eine Datei

Eine Besonderheit im UNIX-Dateisystem ist die Möglichkeit, der eigentlichen Datei mehrere Namen zuzuordnen. Man spricht hier von Links (engl. Verbindungen). Links bieten die Möglichkeit, eine Datei unter ver-

schiedenen Namen auftreten zu lassen oder eine Datei auf einen anderen Datenträger zu legen, als den, auf dem der Zugriff erfolgt.

Der harte Link

Es können beliebig viele Links angelegt werden. Sie können (mit gewissen Einschränkungen) in verschiedenen Verzeichnissen liegen. Der Inhalt der Datei wird erst dann wirklich entfernt, wenn der letzte Link gelöscht wurde. Der Befehl, um einen solchen Link zu erzeugen, lautet:

```
In Originaldatei NeuerName
```

```
In Originaldatei AnderesVerzeichnis
```

Im ersten Fall wird ein neuer Name angegeben, im zweiten Fall wird der gleiche Name in einem anderen Verzeichnis eingetragen. Die Reihenfolge der Parameter entspricht denen von `cp`. Zuerst wird die bereits existierende Quelle genannt und dann das zu erzeugende Zielobjekt. Beispiel: Wenn eine Datei namens **hugo** existiert, bewirkt der Befehl

```
In hugo erna
```

dass hinterher scheinbar zwei Dateien existieren: **hugo** und **erna**. Es ist aber keine Kopie entstanden, sondern beide Einträge zeigen auf die gleiche Datei. Sie können dies leicht überprüfen, indem Sie in der einen Datei eine Änderung durchführen und in der anderen nachschauen. Sie werden die Änderung auch in ihr vorfinden. Sie erkennen, dass für eine Datei ein oder mehrere Links existieren, wenn Sie sich den Dateieintrag mit `ls -l` anschauen. Im Normalfall finden Sie eine 1 zwischen den Dateirechten und dem Besitzer. Das ist die Anzahl der Links, die auf den Dateiinhalt zeigen. Das folgende Beispiel zeigt unsere Dateien **hugo** und **erna**. Als Beispiel für eine normale Datei ist auch **emil** aufgeführt.

```
gaston> ls -l hugo erna emil
-rw-r--r--  1 arnold  users      18 Jul 20 01:51 emil
-rw-r--r--  2 arnold  users      18 Jul 20 01:51 erna
-rw-r--r--  2 arnold  users      18 Jul 20 01:51 hugo
gaston>
```

Man kann an der 2 erkennen, dass auf diese Datei zwei Einträge zeigen. Es ist auch egal, welche Datei zuerst gelöscht wird. Der Dateiinhalt verschwindet erst, wenn der letzte Link gelöscht wird.

Mit dem Parameter `-f` kann ein bestehender gleicher Dateiname überschrieben werden. Beispiel:

```
gaston> ln -f hugo emil
gaston> ls -l hugo erna emil
-rw-r--r--    3 arnold  users      18 Jul 20 01:51 emil
-rw-r--r--    3 arnold  users      18 Jul 20 01:51 erna
-rw-r--r--    3 arnold  users      18 Jul 20 01:51 hugo
gaston>
```

**Nur auf gleichem
Dateisystem**

Der Link funktioniert in dieser Form nicht nur im gleichen Verzeichnis. Die Dateien müssen sich lediglich auf demselben Dateisystem befinden. Das liegt daran, dass die Datei tatsächlich nur einmal da ist. Lediglich der Verzeichniseintrag wird dupliziert. Da aber ein Verzeichniseintrag nur auf eine Datei des gleichen Dateisystems zeigen kann, ist ein plattenübergreifender Link so nicht möglich.

Per Link steuern

Der Link offeriert auch die Möglichkeit, dass ein und dasselbe Programm unterschiedliche Aktionen durchführt, je nachdem wie es heißt. Dieser Effekt wird beispielsweise bei `compress` und `uncompress` benutzt (siehe S. 89). Beide Namen zeigen auf die gleiche Datei. Da ein UNIX-Programm beim Aufruf außer den Parametern auch den Namen erfährt, unter dem es aufgerufen wurde (siehe S. 557), kann `compress` feststellen, ob es unter dem Namen `uncompress` aufgerufen wurde, und setzt dann einfach selbst die Option `-d`.

Ferner ist es möglich, einen Datenbestand von mehreren Benutzern aktuell halten zu lassen, indem jeder Anwender einen Link auf die gleiche Datei hält. Beispielsweise können die `.rhosts`-Dateien, die den Zugriff von fremden Rechnern erlauben (siehe S. 322), auf diese Weise durch eine einzige Datei realisiert werden.

Der symbolische Link

**Greift über
Dateisysteme**

Neben dem harten Link gibt es noch den symbolischen Link. Dieser Link kann auch über Dateisystemgrenzen hinweg greifen. Er ist auch nicht auf Dateien beschränkt: Er kann auch auf ein Verzeichnis zeigen. Allerdings hat er auch einen Haken: UNIX prüft nicht, ob die Datei, auf die der Link zeigt, auch wirklich existiert. UNIX achtet auch nicht darauf, ob jemand eine Datei oder ein Verzeichnis entfernt, auf die bzw. das ein symbolischer Link zeigt.

```
ln -s /etc/printcap myPrintCap
```

Der symbolische Link **myPrintCap** im aktuellen Verzeichnis zeigt auf die Datei **/etc/printcap**. Man kann leicht erkennen, was hinter dem symbolischen Link steckt, wenn man `ls -l` eingibt. Hier zeigt der Link **X11** auf **X11R6**.

Textueller Verweis

```
gaston> ls -l X11
lrwxrwxrwx    1 root    root          5 Mär 17  2001 X11 -> X11R6
gaston>
```

Den symbolischen Link kann man vereinfacht als eine Datei ansehen, die den Ort einer anderen Datei beinhaltet. Damit wird klar, dass der Link auf jede Stelle des UNIX-Verzeichnisbaums zeigen kann, ganz gleich auf welchem Medium es sich befindet. Es ist aber auch einleuchtend, dass die Originaldatei nichts vom Link »weiß« und so entfernt werden kann, obwohl noch ein Link existiert.

Der symbolische Link wird vor allem an zwei Stellen eingesetzt. Zum einen ist er hilfreich, wenn eine Platte am Ende ihrer Kapazität ist. Man kann die Dateien eines gesamten Verzeichnisses oder auch einzelne Dateien auf ein freies Laufwerk auslagern und einen symbolischen Link auf diese Stelle an der Originalposition unterbringen. Das Verschwinden der Datei bleibt für (fast) alle Programme unsichtbar. Der symbolische Link wird auch gern eingesetzt, wenn man die Version einer Software dokumentieren will. Beispielsweise wird das Paket **wollmilchsau** zurzeit in der deutschen Version 1.4 ausgeliefert. Dann installiert man es im Verzeichnis **/opt/wollmilchsau.v.1.4.german**. Um allerdings auf die Software zuzugreifen erstellt man einen symbolischen Link:

```
ln -s /opt/wollmilchsau.v.1.4.german /opt/wollmilchsau
```

Man kann später erkennen, dass die deutsche Version 1.4 installiert wurde. Man kann sogar die Version 1.5 installieren, ohne die Version 1.4 endgültig löschen zu müssen.

Symbolische Links speichern quasi die textuelle Beschreibung der Quelle. Anders ausgedrückt, merken sie sich den Pfadnamen, so wie er beim Anlegen des Links angegeben wurde. Es wird beim Anlegen nicht überprüft, ob der Link wirklich funktioniert. Entsprechend verschwindet der Link auch nicht, wenn das Ziel verschwindet, auf das er verweist. Es ergeben sich darüber hinaus Stolperfallen, wenn nicht ein absoluter, sondern ein relativer Pfad als Ziel eines symbolischen Links angegeben wird. Im folgenden Beispiel liegt der zu erzeugende Link in einem Unterverzeich-



nis, in diesem Fall in **savedir/my20020105**. Da aber das Quellverzeichnis relativ zur aktuellen Position steht, wird auch der relative Pfad abgelegt. Aus Sicht des symbolischen Links liegt aber das Verzeichnis **my** nicht in **../my**, sondern in **../../my**. Es gibt keine Fehlermeldung beim Erzeugen des Links, aber die erste Benutzung wird scheitern.

```
ln -s ../my savedir/my20020105
```

1.7.5 Besondere Dateien

Mit den Verzeichnissen und den Links sind bereits zwei Arten besonderer Dateien vorgestellt worden. Bei einem `ls -l` kann man am ersten Buchstaben links ein Verzeichnis an einem `d` für `directory` erkennen. Einen symbolischen Link erkennt man an einem `l`. Es gibt noch weitere besondere Dateien.

```
lrwxrwxrwx   1 root    root      5 Mär 17  2001 X11 -> X11R6
drwxr-xr-x   9 root    root     4096 Jan  5 19:34 X11R6
```

special files zeigen
auf die Peripherie

Es gibt so genannte special files, also spezielle Dateien, die einen Zugriff auf Peripheriegeräte ermöglichen. Man findet sie im Verzeichnis **/dev**. Sie tragen ein kleines `c` oder `b` als Kennzeichen (siehe S. 158). Man kann diese Direktzugriffe auf die Peripherie mit den normalen Dateizugriffen durchführen, sollte dies im Normalfall aber vermeiden.¹⁷ Im Allgemeinen gibt es Programme, die den Zugriff steuern. So wird der Zugriff auf den Drucker (**/dev/lp0**) üblicherweise über das Programm `lp` oder `lpr` geregelt. Das verhindert, dass sich verschiedene Benutzer gegenseitig die Ausdrücke durcheinander bringen.

Sockets und Pipes

Dateien mit einem `s` sind so genannte Sockets, und Dateien mit einem `p` sind named pipes. Beide dienen zur Kommunikation zwischen Programmen. Diese Spezies sollte man im Normalfall in Ruhe lassen. Die sie benutzenden Programme könnten erheblich durcheinander geraten, wenn sie plötzlich fehlen oder manipuliert würden.

1.7.6 Der Dateityp: file

Der Befehl `file Datei` zeigt an, welchen Typ eine Datei hat. Hierbei können insbesondere folgende Fälle auftreten:

¹⁷ Da nicht jeder Anwender das Talent hat, sich in Zurückhaltung zu üben, stärkt ihn UNIX in Momenten der Schwäche durch entsprechende Rechtevergabe.

Bezeichnung	Inhalt
executable oder object	Das sind die übersetzten Programme.
ascii text	Normaler Text, den man sich im Editor ansehen kann
directory	Verzeichnis
commands text	Shellskripten oder Ähnliches
symbolic link	Symbolischer Link mit Zielangabe
alias	Namensgebung der Shell (siehe S. 99)

Tabelle 1.8 Dateitypen

Daneben gibt es noch einige Standardtypen, die file erkennen kann, wie beispielsweise Bibliotheken. Die Informationen über den Typ bekommt file aus der Datei `/etc/magic` bzw. aus `/var/share/misc/magic` bei FreeBSD. In dieser Datei stehen Kennzahlen und Positionen, um bestimmte Dateigruppen eindeutig zu identifizieren.

Kennungen von Dateitypen

1.8 Zugriff auf mehrere Objekte

Will man mehrere Dateien als Argumente für einen UNIX-Befehl verwenden, kann man sie meist einfach aufzählen. Die meisten Kommandozeilenprogramme sind unter UNIX so geschrieben, dass sie beliebig viele Dateien als Argumente akzeptieren. Um die Aufzählung zu vereinfachen, kann man mit Hilfe einer Maske mehrere Dateien zusammenfassen. Die Shell sucht die auf die Maske passenden Dateien zusammen und übergibt sie dem aufgerufenen Programm als Liste.

Die Shell löst Wildcards auf.

1.8.1 Wildcards: *, ? und die eckigen Klammern

Um Dateien mit ähnlichen Namen zu ermitteln, wird meist der Stern als Platzhalter verwandt. Dieser steht als Ersatz für beliebig viele Zeichen. An der Stelle des Sterns kann auch gar kein Zeichen stehen. Die folgende Liste zeigt einige Beispiele:

Stern

- ▶ `ls prog*` – Alle Dateien, die mit `prog` anfangen.
- ▶ `ls *mein` – Alle Dateien, die mit `mein` aufhören.
- ▶ `ls OS*.c` – Alle Dateien, die mit `OS` anfangen und mit `.c` aufhören.
- ▶ `ls *dat*` – Alle Dateien, die `dat` im Namen enthalten.

Soll eine genaue Anzahl von Zeichen freigehalten werden, benutzt man das Fragezeichen. Es steht für genau ein Zeichen. `M??s` steht also für

Fragezeichen

Maus, Mais oder Muks. Murks würde nicht passen, da die drei Buchstaben zwischen M und s nicht auf die zwei Fragezeichen passen. Auch eine Kombination aus Fragezeichen und Sternen macht Sinn. Wenn man beispielsweise alle Dateien und Verzeichnisse, die mit einem Punkt beginnen, löschen möchte,¹⁸ sollte man lieber nicht `rm -r .*` eingeben. Da der Stern auch die leere Zeichenkette symbolisiert, würde damit das aktuelle und das Elternverzeichnis ausgeräumt. Das Elternverzeichnis ist das davor liegende Verzeichnis und wird mit `..` angesprochen. Besser ist da der Gedanke, mit `rm -r .??*` zu arbeiten. Damit werden dann weder `.` noch `..` getroffen, da beide nicht aus drei Zeichen bestehen. Allerdings würde auch die Datei `.ab` stehen bleiben.

Eckige Klammern Neben diesen bekannten Wildcards gibt es noch die Möglichkeit, mit den rechteckigen Klammern für ein Zeichen gewisse Alternativen zu verwenden. Beispielsweise bedeutet `[Mm]akefile`, dass die Datei **makefile** oder **Makefile** heißen kann. Die Datei `[A-Z][0-9]?*` muss mit einem Großbuchstaben beginnen, darauf muss eine Ziffer folgen, und sie muss mindestens aus drei Zeichen bestehen.

1.8.2 Sonderzeichen als Parameter

Will man ein Zeichen an ein Programm geben, das von der Shell interpretiert wird, muss es vor ihr geschützt werden. Dies kann bei einzelnen Zeichen durch Voranstellen eines Backslash (`\`) erfolgen. Die deutsche Übersetzung von Backslash hieße »rückwärtiger Schrägstrich«. `*` steht für einen Stern, `\?` für ein Fragezeichen im Namen. Man sollte, um Missverständnisse zu vermeiden, jedoch diese Zeichen in Dateinamen besser nicht benutzen.

Alternativ kann das Argument auch in Anführungszeichen (`"`) oder Hochkommata (`'`) gesetzt werden. In diesem Fall interpretiert die Shell nicht die Sonderzeichen, sondern reicht sie direkt an das aufgerufene Programm weiter.¹⁹

1.9 Editoren

Standardeditoren Da unter UNIX fast alle Konfigurationen über Textdateien laufen, ist ein guter Editor ein wichtiges Werkzeug. Unter UNIX gibt es einmal `vi`, einen **vi und emacs**

¹⁸ Es ist übrigens keine gute Idee, diese Dateien aus dem Heimatverzeichnis löschen zu wollen. Sollten Sie es gerade getan haben, vergessen Sie, wo Sie das gelesen haben.

¹⁹ Der Unterschied zwischen beiden ist, dass in Anführungszeichen Variablen noch aufgelöst werden, in Hochkommata nicht (siehe S. 486).

textorientierten Editor, der normalerweise auf jeder Maschine installiert ist. Der zweite bekannte Editor ist emacs, der durch seine Programmierbarkeit erweiterbar und so extrem leistungsfähig ist. Der emacs ist sehr weit verbreitet, aber nicht auf jeder Maschine standardmäßig installiert.

In Zeiten der grafischen Oberflächen wirkt die Bedienung auf manchen Einsteiger etwas archaisch. vi und emacs existieren aber nicht nur aus Gewohnheit auf UNIX-Systemen, sondern weil sie gerade durch ihr ungewöhnliches Konzept auch ungewöhnlich leistungsfähig sind. Zugegebenermaßen erschließen sich die Fähigkeiten dieser Werkzeuge nicht auf Anhieb. Allein die Tatsache, dass vi und emacs auch dann funktionieren, wenn die Terminalemulation die Funktionstasten nicht interpretieren kann, macht diese Programme wertvoll für Administratoren, die manchmal auf bizarren Wegen in eine notleidende Maschine eindringen.

Wer seine UNIX-Maschine nur von einer grafischen Oberfläche wie dem X Window System aus bedient, findet im Normalfall auch jede Menge Editoren, die sich vielleicht intuitiver bedienen lassen. Das fängt mit xedit an, der jeder X-Installation beiliegt. vi und emacs sind allerdings extrem leistungsfähige Editoren, insofern lohnt sich auf lange Sicht die Einarbeitung.

1.9.1 vi

Der Editor vi ist sehr verbreitet. Man findet ihn nicht nur auf jeder UNIX-Maschine, sondern auch auf diversen anderen Plattformen. Neben oder statt des originalen vi werkelt oft ein Klon, der nicht selten einige Erweiterungen erfahren hat. Die bekanntesten Vertreter sind elvis und vim.

vi hat ein originelles Konzept, dessen Vorteile nicht auf Anhieb erkennbar sind. Es gibt drei Modi, in denen sich das Programm befinden kann: den Kommandomodus, den Eingabemodus und den Befehlsmodus.

Nach dem Starten des vi befindet man sich im Kommandomodus. Hier kann man im Text positionieren, suchen und Kommandos zur Textveränderung geben. Durch solche Kommandos gelangt man in den Eingabemodus.

Der Eingabemodus ist recht simpel. Man kann Text eingeben und mit der Korrekturtaste (Backspace) die letzten Vertipper innerhalb der aktuellen Zeile korrigieren. Alle anderen Korrekturen und Maßnahmen werden vom

Arbeiten auch
ohne
Funktionstasten

Kommandomodus

Eingabemodus

Kommandomodus aus erledigt, in den man durch die Taste ESC²⁰ wieder zurückkehrt.

Befehlsmodus Vom Kommandomodus aus erreicht man durch Eingabe des Doppelpunktes den Befehlsmodus. Der Cursor geht in die unterste Zeile und erwartet einen durch Return abgeschlossenen Befehl. Diese Befehle entsprechen übrigens zum Teil denen des sed (siehe S. 79). Sie betreffen in erster Linie den Text als Ganzes, beispielsweise das Sichern und Wechseln von Texten.

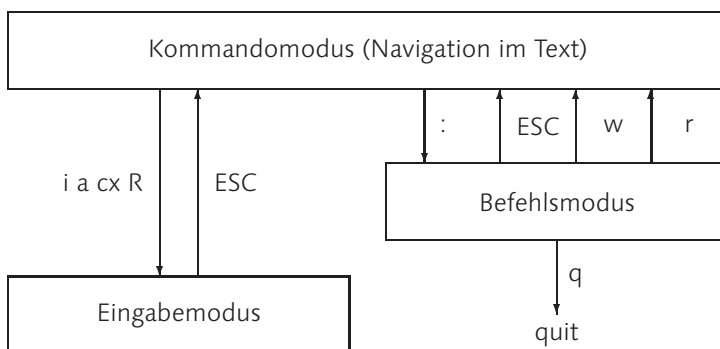


Abbildung 1.3 Die Modi des vi

Abbildung 1.3 zeigt die Modi des vi im Überblick. Nachdem man den vi gestartet hat, befindet man sich im Kommandomodus, in dem auch durch den Text navigiert wird. Cursorpositionierung, Blättern und Suchen findet hier statt. Mit Kommandos wie `i` (insert), `a` (append), `cx`²¹ (change) und `R` (replace) gelangt man an der aktuellen Position in den Eingabemodus. Dort gibt man Text ein. Ist die Eingabe abgeschlossen, kehrt man mit `ESC` wieder in den Kommandomodus zurück, um zur nächsten Stelle zu navigieren. Vom Kommandomodus gelangt man mit dem Doppelpunkt in den Befehlsmodus. Hier gibt man unter anderem die Befehle zum Verlassen, zum Sichern und Hinzuladen.

Verlassen

Das Wichtigste zuerst: Man kommt aus vi heraus, indem man zunächst ggf. den Editiermodus verlässt. Das geschieht durch `ESC`. Falls ein `ESC` zu viel gedrückt wurde, gibt vi ein akustisches Signal von sich.

²⁰ Man kann darüber streiten, ob diese Taste als Abschluss der Eingabe eine glückliche Wahl war. Inzwischen verwenden fast alle Programme die `ESC`-Taste, wenn sie Eingaben verwerfen wollen. Unter vi jedenfalls wird die Eingabe bestätigt und der Modus verlassen.

²¹ Das `x` steht hier für ein Zeichen, das den Bereich bestimmt, auf den die Änderung wirken soll.

Dann wechselt man mit dem Doppelpunkt in den Befehlsmodus. Der Cursor geht in die letzte Zeile. Als Befehl gibt man ein einfaches, kleines q (quit) ein und bestätigt mit der Returnntaste. Sollte vi nörgeln, dass Änderungen noch nicht gesichert sind, kann man dies mit :q! ignorieren oder mit :wq (write quit) die Änderungen vor dem Verlassen sichern.

Befehl	Wirkung
ESC :q	verlassen von vi
ESC :q!	verlassen von vi und verwerfen aller Änderungen
ESC :wq	verlassen von vi und vorheriges Sichern unter dem bisherigen Namen

Tabelle 1.9 wie man vi verlässt

Starten des vi

Wollen Sie eine Datei erstellen, geben Sie vi, gefolgt von dem Namen der neuen Datei, an. Sie landen in einem Bildschirm, auf dem alle Zeilen mit einer Tilde beginnen. Am Ende des Bildschirms stehen der Name der Datei und die Position. Das Tildezeichen soll anzeigen, dass das Ende des Textes erreicht ist. Um etwas einzugeben, tippen Sie ein kleines i (insert). Sie sehen an der Statuszeile, dass vi in den Einfügemodus gewechselt ist. Sie können nun einige Zeilen Text eingeben, Tippfehler sollten Sie zunächst ignorieren. Ist der Text eingegeben, erreichen Sie mit ESC wieder den Kommandomodus. Nun sollten Sie den Text sichern, indem Sie mit einem Doppelpunkt in den Befehlsmodus wechseln. Geben Sie w (write) ein, und der Text wird gesichert. Als Dateiname wird derjenige verwandt, den Sie beim Start von vi angaben. Man kann nun vi wieder mit :q verlassen.

Die Arbeitsdatei
beim Start nennen

Sie können die Datei wieder bearbeiten, indem Sie den vi wieder mit dem Dateinamen starten. Nun sehen Sie den Text im Überblick und befinden sich wieder im Kommandomodus. Zunächst bewegen Sie sich an die Stelle, wo Sie etwas ändern wollen. Mit + und - können Sie zeilenweise auf und ab gehen. Mit der Leertaste und Backspace können Sie nach links und rechts in der Zeile über den Text fahren. Auf vielen Systemen arbeiten auch die Cursortasten. Haben Sie die Zielstelle erreicht, können Sie wieder mit i neuen Text hinzufügen oder mit dem kleinen x (wie extract oder wie eine kleine Schere) einzelne Zeichen unter dem Cursor löschen. Zwei Zeilen kann man mit einem großen J (join) zusammenfügen. Mit diesen Kommandos kommt man bereits sicher durch den vi, auch wenn das noch nicht sehr komfortabel ist.

Grundlegende
Navigation

Bewegungen durch den Text im Kommandomodus

Auf den meisten Systemen funktionieren die Standardtasten für die Cursorbewegungen. Das wird den meisten Anwendern am einfachsten erscheinen. Leider funktioniert das im Allgemeinen nur auf dem lokalen Rechner. Sobald man mit Hilfe des Netzes auf fernen Rechnern arbeitet (siehe telnet, S. 320), hat man es mit Terminalemulationen zu tun, bei denen oft die Funktionstasten nicht mehr funktionieren. In solchen Fällen arbeiten aber immer noch die normalen Tasten und die ctrl-Kombinationen. Insofern ist es hilfreich, dass vi auch mit diesen Tasten bedienbar ist.

Weite	zurück	vor
ein Zeichen	h oder Backspace	Leertaste
ein Wort	b	w
Zeilenanfang/-ende	^	\$
an die n-te Spaltenposition		n
Zeile	-	+
Seite	ctrl-F	ctrl-B
zur Zeile n	nG	
Textanfang/-ende	1G	G

Tabelle 1.10 Cursorbewegungen im vi

Suchen im Text

Suchen: / vi sucht mit dem Schrägstrich. Daraufhin geht der Cursor in die unterste Zeile. Hier gibt man den Suchbegriff ein und bestätigt mit Return. Um das nächste Erscheinen des gleichen Begriffs zu suchen, gibt man ein kleines n ein. Mit einem großen N kommt man zum vorangegangenen Erscheinen des Begriffs. Will man von vornherein rückwärts suchen, verwendet man das Fragezeichen statt des Schrägstrichs als Befehl. Auch hier arbeiten n und N, allerdings jetzt in der anderen Richtung.

Editieren

Um den Text zu bearbeiten, wird man zunächst Text einfügen. Dazu gibt es folgende Kommandos:

Taste	Wirkung
i	einfügen (engl. insert) an der Cursorstelle
I	einfügen vor dem ersten Wort der Zeile
a	hinter der Cursorstelle einfügen (engl. append)
A	an das Ende der Zeile anhängen

Tabelle 1.11 Kommandos zum Wechsel in den Editiermodus

Die Eingaben werden durch ESC bestätigt. Man kann, anstatt Text einzufügen, auch existierenden Text ersetzen. Dabei dient das c als Zeichen für Change. Es folgt ein Zeichen, das angibt, was man ersetzen will. Hier heißt w Wort, \$ bis Zeilenende, l ein Zeichen, und die Verdoppelung des c steht für die ganze Zeile. Auf dem Bildschirm wird der Ausschnitt, den man ersetzt, durch ein Dollarzeichen begrenzt.

Tasten	Wirkungsbereich
cw	Wort
c\$	bis Ende der Zeile
cc	aktuelle Zeile
cl oder r	Zeichen
R	geht in den Überschreibmodus bis zum ESC

Tabelle 1.12 Ändern

Löschen und Kopieren

Das Kommando zum Löschen von Textteilen beginnt mit d (für engl. delete). Wie beim Ändern wird die Reichweite des Löschens durch den zweiten Buchstaben bestimmt, dabei gilt w für ein Wort, \$ bis zum Ende der Zeile, und die Verdoppelung des Kommandobuchstabens betrifft die komplette Zeile. Für den Buchstaben wird das l verwendet, aber auch hier gibt es einen speziellen Buchstaben.

Tasten	Wirkungsbereich
dw	Wort
d\$	bis Ende der Zeile
dd	aktuelle Zeile
dl oder x	Zeichen

Tabelle 1.13 Löschkommandos unter vi

Löschen bis Suchkriterium

Man kann durch Verwendung der Suchfunktion das Ziel angeben, bis zu dem gelöscht werden soll. Man startet an der aktuellen Cursorposition. Beispiel: d/abc plus Return-Taste löscht alles zwischen hier und dem nächsten Auftreten der Zeichenkette abc.

Der Puffer

Die gelöschten Textbereiche werden in einem Puffer zwischengelagert. Dieser Text kann jederzeit an jeder Stelle mit p oder P wieder aus dem Puffer geholt werden. Dabei fügt p hinter und P an der aktuellen Cursorposition ein. Man kann den Puffer allerdings auch füllen, ohne das Original zu löschen. Man verwendet lediglich das Kommando y (yank) statt d.

Tasten	Wirkungsbereich
yw	Wort
y\$	bis Ende der Zeile
yy	aktuelle Zeile

Tabelle 1.14 Zwischenspeichern

Zeilenvorschub löschen

J vereinigt Zeilen

Um zwei Zeilen aneinander zu hängen, also den Zeilenvorschub der oberen Zeile zu löschen, geht man in die obere Zeile und gibt ein großes J (join) ein. Die beiden Zeilen werden vereinigt, und dazwischen steht ein Leerzeichen.

Geklammerte Bereiche

% als Bereich zwischen Klammern

Für Programmierer besonders interessant ist der Umgang mit Klammern. Man positioniert den Cursor auf einer Klammer, und vi springt durch das %-Zeichen auf die korrespondierende Klammer. Dabei werden runde, eckige und geschweifte Klammern akzeptiert. Das Prozentzeichen kann man auch als Bereichsangabe anderer Befehle verwenden. Die Tasten-

kombination d% löscht den Bereich von der aktuellen Klammer unter dem Cursor bis zur korrespondierenden Klammer.

Besonders interessant ist das Einrücken in Verbindung mit Klammerungen für C-Programmierer. Man kann sich auf eine geschweifte Klammer stellen und durch das Kommando >% den kompletten Block um so viele Leerzeichen einrücken, wie in der Variablen shiftwidth definiert ist.

Einrücken

Tasten	Wirkung
%	springt von dieser Klammer zur korrespondierenden Klammer
d%	löscht alles zwischen dieser und der korrespondierenden Klammer
c%	ersetzt alles zwischen dieser und der korrespondierenden Klammer
y%	schiebt alles zwischen den Klammern in den Puffer
<% bzw. >%	Ein- bzw. Ausrücken des eingeklammerten Bereichs

Tabelle 1.15 Klammern

Mehrfachausführung

Soll ein Kommando mehrfach ausgeführt werden, stellt man einem beliebigen Kommando die Häufigkeit voran. Beispiele:

Kommando	Wirkung
4dw	löscht die nächsten vier Wörter
5cw	ersetzt die nächsten fünf Wörter durch die Eingabe bis zum ESC
77i - ESC	erzeugt 77 Bindestriche
3cl	ersetzt die nächsten drei Zeichen durch die Eingabe
12yy	kopiert die nächsten zwölf Zeilen in den Kopierpuffer

Tabelle 1.16 Beispiele für Mehrfachausführung

Mach's nochmal, vi!

Ein besonderes Feature von vi ist der Punkt. In der Kommandoebene eingegeben, wiederholt er das letzte Kommando an der aktuellen Cursorposition. Auch vor dem Punkt kann man die Häufigkeit angeben. Hat man nach 10dd noch 10. getippt, werden 100 Zeilen gelöscht.

Ein Punkt wiederholt das letzte Kommando

Schief gelaufen

Wenn mal etwas schief ging: Versehentliches Löschen und andere ungewollte Vorgänge kann man mit u (undo) ungeschehen machen. Befindet

man sich in einem unbekannten Modus, erreicht man mit ESC die Kommandoebene, von der aus das u eingegeben werden muss.

**Undo nicht
unbedingt
mehrstufig**

Stellt man fest, dass nicht der letzte, sondern der vorletzte Befehl den Schaden verursacht hat, der 100 Seiten Manuskript mit der Schredderfunktion bearbeitete, kann man zwei Dinge tun. Das erste ist, dass man u ein weiteres Mal drückt und hofft, eine der neueren Implementationen von vi zu benutzen, die ein Undo in mehreren Stufen erlauben. Es kann aber auch sein, dass statt der mehrstufigen Änderung nur ein Redo erfolgt ist. Dann sollte man aufstehen, tief durchatmen und überlegen, in welchem Zustand sich die Datei auf der Platte befindet und dann den vi mit :q! verlassen, um die letzte Version zu verwenden oder das Thema Zwischenspeichern noch einmal intensiv durchdenken.

Ein versierter vi-Benutzer würde an dieser Stelle allerdings den aktuellen Text nicht mit :q! verwerfen, sondern ihn unter einem anderen Namen sichern. Dazu wird :w, gefolgt von einem Leerzeichen und dem neuen Dateinamen, aufgerufen. Also sichert der Befehl :w datei den Text unter dem Namen **datei**. Anschließend kann man mit dem Kommando diff (siehe S. 553) ermitteln, welche Unterschiede zwischen der Datei **datei** und dem letzten Stand bestehen.

Sollte stattdessen die verwendete Version ein mehrstufiges Undo beherrschen, stellt sich die Frage, wie man ein Undo zu viel wieder rückgängig machen kann. Hier empfiehlt sich ein Experiment mit ctrl-R.

Wichtige Befehle des Befehlsmodus

sichern und laden

Der Befehlsmodus wird mit dem Doppelpunkt aktiviert. Die Kommandos q zum Verlassen und w zum Sichern wurden bereits vorgestellt. Auch dass man dem Kommando w als Parameter noch einen Dateinamen mitgeben kann, um den Text unter diesem Namen zu speichern, wurde bereits erwähnt. Als Gegenstück zum w gibt es ein r (read), um Texte in den aktuellen Text hineinzulesen. Auch hier wird der Dateiname als Parameter angegeben. Der angegebene Text erscheint danach an der aktuellen Cursorposition.

Shellaufruf

Mit dem Ausrufezeichen wird die restliche Befehlszeile an eine Shell weitergeleitet. Man kann so aus dem Editor einen Compiler starten oder andere Systemkommandos geben. Probieren Sie einmal :!ls aus!

Befehl	Wirkung
:n	Lädt die nächste Datei, falls vi mit mehreren Dateien gestartet wurde
:r <i>Datei</i>	Lädt die <i>Datei</i> an der Cursorposition hinzu
:d	Löscht Zeile
:! <i>Kommando</i>	Startet eine Shell, um das Kommando abzusetzen

Tabelle 1.17 Wichtige Befehle

Für viele Kommandos der Befehlszeile kann ein Bereich von Zeilen vorangestellt werden, auf den sie sich auswirken sollen. So bewirkt das Voranstellen einer 8 vor w datei, dass Zeile 8 unter dem Namen datei gespeichert wird. Einen Bereich von Zeilen gibt man durch die erste Zeilennummer, gefolgt von einem Komma und der Zahl der letzten Zeile, an. 12,15 bezeichnet also die Zeilen zwischen 12 und 15. Für bestimmte Zeilen gibt es besondere Symbole. Die aktuelle Zeile, in der sich der Cursor befindet, wird durch einen Punkt bezeichnet. Das Dollarzeichen bezeichnet die letzte Zeile. Innerhalb der Bereichsangabe darf kein Leerzeichen stehen.

Befehle beginnen mit dem Zeilenbereich

Angabe	Bereich
25	die Zeile 25
5,25	der Bereich von Zeile 5 bis 25
.,25	der Bereich von der aktuellen Zeile bis Zeile 25
25,\$	von Zeile 25 bis Dateiende
.,\$	von der aktuellen Zeile bis zum Dateiende

Tabelle 1.18 Bereichsangaben

Ersetzen

Um eine Zeichenfolge durch eine andere zu ersetzen, muss man zunächst mit einem Doppelpunkt in den Befehlsmodus wechseln. Der folgende Befehl bewirkt das Ersetzen der Zeichenkette »suchmuster« durch die Zeichenkette »ersetze« im gesamten Text.²²

²² Dies ist mein Lieblingsbefehl. Nennen Sie diese Folge von Buchstaben auf die Frage nach dem Ersetzungsbefehl von vi mit der Anmerkung »ist doch eigentlich nahe liegend, oder?« und man wird Ihren Namen nur noch mit Ehrfurcht aussprechen.

:1,\$ s/suchmuster/ersetze/g

Dieser Befehl sollte zunächst am Leerzeichen in zwei Teile zerlegt werden. Der Bereich, in dem der Befehl wirkt, wird gleich zu Anfang der Zeile angegeben. Steht dort nichts, wird nur die aktuelle Zeile herangezogen. In diesem Fall ist 1,\$ der Bereich von der ersten bis zur letzten Zeile. Das Dollarzeichen ist im vi immer ein Symbol für das Ende.

Dann folgt der Befehl s für suchen (engl. search). Das Suchmuster und die Ersetzung müssen voneinander und von den Kommandos vorher und nachher abgetrennt werden. Dazu wird meist der Schrägstrich verwendet. Es könnte auch jedes andere Zeichen sein, das im Suchmuster nicht vorkommt. Am Ende kommt das g (global), damit die Ersetzung nicht beim ersten Mal stehen bleibt.

Das Suchmuster ist übrigens ein regulärer Ausdruck, so wie er auch von grep verwendet wird (siehe S. 84).

Settings

Um Optionen ein- oder abzuschalten, geht man zunächst mit dem Doppelpunkt in den Befehlsmodus und gibt dort set, gefolgt von der Option, an. Man kann vi durch viele Optionen konfigurieren. Die verschiedenen Implementationen bieten da unterschiedliche Möglichkeiten. Um eine Übersicht zu erhalten, gibt man set all an und blättert mit der Leertaste durch die Seiten. Mit ESC kommt man in den Kommandomodus zurück.

Einschalten	Ausschalten	Wirkung
set all		zeigt die aktuellen Settings an
set list	set nolist	zeigt Sonderzeichen an
set sw <i>n</i>		(shiftwidth) setzt Einrücklevel auf <i>n</i> Stellen
set ai	set noai	(autoinsert) Automatisches Einrücken
set nu	set nonu	(number) Zeilennummern anzeigen
set smd	set nosmd	(showmode) zeigt den Textmodus an
set sm	set nosm	(showmatch) Anzeige der Gegenklammer bei Eingabe

Tabelle 1.19 Schalter

Steuerung des Zeilenumbruchs

Ist eine Zeile länger als die Bildschirmbreite, dann zeigt vi den Rest in der nächsten Zeile an. Dieses Verhalten kann in manchen Fällen sehr lästig sein. Durch nowrap schaltet man es ab, mit wrap wieder ein. Bei der Texteingabe wird erst bei Return eine neue Zeile erzeugt. Dieses kann

man durch `wm` (wrapmargin) ändern. Gibt man eine Zahl ungleich 0 an, wird die entsprechende Anzahl Zeichen vor dem Erreichen des rechten Rands umbrochen. Dabei nimmt `vi` den Umbruch immer wortweise vor. Er nimmt also das angefangene Wort mit in die neue Zeile und fügt davor ein Zeilenumbruchzeichen (new line) ein.

1.9.2 emacs

emacs existiert als Textversion oder als xemacs für das X Window System. Da emacs unter der GPL (Gnu Public License) steht, gibt es vermutlich auf beinahe jeder Plattform eine Implementation. Der Autor Richard Stallman gilt als der Vater der Open Source-Bewegung.

emacs ist extrem erweiterbar. Das Informationssystem `info`, das bereits erwähnt wurde (siehe S. 28), basiert auf emacs. Für emacs gibt es diverse Anpassungen für das Syntax-Highlighting, also für die farbigen Hervorhebung der Elemente verschiedener Programmiersprachen.

Zur Schreibweise der Kommandos: Statt `ctrl-x` oder `Strg-x` schreibt man in der emacs-Dokumentation `C-x`. Als weitere wichtige Taste wird die Metataste gebraucht. Sie wird oft mit einer Raute gekennzeichnet, oder es wird die Alt-Taste bei PCs verwendet. Die Metataste wird mit `M` abgekürzt, `Meta-k` schreibt man also in der emacs-Dokumentation `M-k`. Auch wenn es in der Dokumentation von emacs durchgehend in der genannten Form notiert wird, wird mir Richard Stallman sicher vergeben, wenn ich aus bei der Schreibweise `ctrl-` und `meta-` bleibe, um eine einheitliche Schreibweise in allen Kapiteln zu bewahren.

Tasten-
schreibweise

Verlassen

Das Wichtigste zuerst: Man kommt aus dem Emacs durch die Kombination `ctrl-X ctrl-C` (exit cancel) wieder heraus.

`ctrl-X ctrl-C`

Tutorial aufrufen

Nach der Tastenkombination `ctrl-H t` startet ein sehr gelungenes Tutorial. Mit diesem kann man sich gut in den emacs einarbeiten.

Bewegungen im Text

Die Bewegungen im Text erfolgen durch Kontrollzeichen. Meistens funktionieren auch die Cursortasten. Da diese aber leicht bei Sitzungen per telnet oder Ähnlichem versagen, ist es gut zu wissen, wie man sich mit diesen Kommandos bewegt.

Weite	zurück	vor
Ein Zeichen	ctrl-B	ctrl-F
Ein Wort	meta-B	meta-F
Zeilenanfang/-ende	ctrl-A	ctrl-E
Satz	meta-A	meta-E
Textanfang/-ende	meta-<	meta->

Tabelle 1.20 Cursorbewegung

Suchen im Text

ctrl-S für
inkrementelles
Suchen

emacs verwendet das inkrementelle Suchen. Man gibt ctrl-S für das Suchen ein und gibt den Suchbegriff Zeichen für Zeichen ein. Dabei positioniert emacs immer auf das nächste Vorkommen des bisher eingegebenen Wortes. Durch die Return-Taste wird die Suche beendet. Mit nochmaligem ctrl-S wird das nächste Vorkommen der gleichen Zeichenfolge angesprungen. Mit ctrl-R statt ctrl-S wird die Suche rückwärts gestartet.

Löschen

Taste	Löschbereich
ctrl-D	Zeichen
meta-D	Wort
ctrl-K	bis Ende der Zeile
meta-K	bis zum Ende des Satzes

Tabelle 1.21 Löschen

Größere Textbereiche, also nicht einzelne Zeichen, werden in einem Puffer zwischengelagert. Der Inhalt dieses Puffers kann jederzeit an jeder Stelle mit ctrl-Y wieder hervorgebracht werden.

Mehrfachausführung

ctrl-U Zahl Befehl

Soll ein Kommando mehrfach ausgeführt werden, stellt man die Häufigkeit voran. Damit emacs merkt, dass es sich um eine Wiederholung und nicht etwa um eine Zahleneingabe handelt, wird der Zahl wiederum ein ctrl-U vorangestellt. Beispiele:

Tastenkombination	Wirkung
ctrl-U 4 meta-D	löscht die nächsten vier Wörter
ctrl-U 77 -	erzeugt 77 Bindestriche

Tabelle 1.22 Beispiele für die Mehrfachausführung

Schief gelaufen

Wenn mal etwas schief ging: Versehentliches Löschen und andere ungewollte Vorgänge kann man mit `ctrl-X u` oder noch einfacher mit `ctrl-_` ungeschehen machen. Will man eine sanfte Unterbrechung eines Vorganges von emacs haben oder hat man versehentlich bei der Mehrfachausführung zu hohe Werte angegeben, kann man mit `ctrl-G` emacs wieder in den Normalzustand bringen.

Befindet sich emacs in einem rekursiven Editierlevel, kann man mit dreimaligem Drücken von ESC wieder in den Normalzustand gelangen.

Fenster

emacs kann den gleichen Text in mehreren Fenstern anzeigen. Mit Hilfe von `ctrl-X 2` wird der Bildschirm vertikal, durch `ctrl-X 3` wird er horizontal geteilt. Man reduziert alle Fenster auf eines, indem man `ctrl-X 1` angibt.

1.10 UNIX-Kommandos verknüpfen

UNIX-Kommandos wirken manchmal etwas spartanisch. Die Ausgaben der Kommandos beschränken sich auf die jeweiligen Aufgaben. Während `ls` wirklich nur die Dateinamen anzeigt, zeigt der analoge Befehl `dir` unter MS-DOS gleich noch den Namen der Platte, die Anzahl der Dateien und den noch freien Platz auf der Festplatte. Der Grund dafür liegt nicht in erster Linie in der Faulheit der UNIX-Programmierer, sondern darin, dass es zur UNIX-Philosophie gehört, Programme zu kombinieren, um die gewünschten Informationen zu erhalten. Die Daten, die ein Befehl erzeugt, sollten so gestaltet sein, dass sie von einem anderen Programm weiterverarbeitet werden können.

Jedes Programm erledigt genau seine Aufgabe

Um auf das Beispiel mit der Anzahl der Dateien zurückzukommen, würde man unter UNIX das Ergebnis von `ls` durch den Befehl `wc` (wordcounter; das Programm zählt Wörter, siehe S. 78) bearbeiten lassen und erhielte damit die Anzahl der aufgelisteten Dateien. Das Kommando dazu sieht so aus:

Programme kombinieren

ls | wc

Als Kupplung zwischen den Befehlen wird die Ein- und Ausgabe der Programme verwendet. Unter UNIX kann beides leicht umgeleitet und aneinander gehängt werden. Hier wird die Ausgabe von `ls` in die Eingabe von `wc` geleitet. Der senkrechte Strich soll eine Röhre (engl. pipe) darstellen.

1.10.1 Ein- und Ausgabe als Datenstrom

Ein- und Ausgabe
sind Dateien

Die Tastatureingabe und die Bildschirmausgabe werden unter UNIX jeweils als Datei aufgefasst. Die Standardeingabe heißt **stdin** und die Ausgabe **stdout**. Um von der Tastatur ein Dateiende simulieren zu können, wird die Tastenkombination `ctrl-D` am Anfang der Zeile verwendet.

`ctrl-D` ist die
Dateiende-
kennung

Das `ctrl-D` ist bereits vom Abmelden her bekannt. Tatsächlich erwartet die Shell aus der Standardeingabedatei ihre Befehle und beendet sich, wenn diese Datei beendet wird. Dieses Verhalten kann man natürlich dazu nutzen, mehrere Shellbefehle in eine Datei zu schreiben und sie am Stück ausführen zu lassen. Dazu ruft man die Shell mit dem Kommando `sh` und dieser Datei als Parameter auf. Die Zusammenstellung mehrerer Befehle ist bereits ein einfaches Programm. Abläufe, die immer gleich sind, kann man so leicht automatisieren. Mit solchen Programmen wird sich das Kapitel über die Shellprogrammierung (siehe S. 479) befassen.

stderr: Kanal für
Fehlermeldungen

Wie in den folgenden Abschnitten zu sehen ist, kann man das **stdout** des einen Programms und das **stdin** eines anderen Programms miteinander koppeln. Das zweite Programm verarbeitet also die Ausgaben des ersten Programms als seine Eingabe. Da Fehlermeldungen in diesem Datenstrom wenig hilfreich sind, gibt es neben **stdout** noch einen separaten Ausgabekanal für die Fehlermeldungen namens **stderr**. Normalerweise zeigt auch dieser auf das Terminal. Auf diese Weise kann man beliebig viele Programme aneinander hängen, ohne sich allzu große Sorgen darüber machen zu müssen, was passiert, wenn eine Komponente einen Fehler erzeugt. Die Fehlermeldung erscheint dennoch auf der Konsole, während im Datenstrom nur gültige und verarbeitbare Daten fließen.

1.10.2 Umleitung

Durch Anhängen von `< meinedatei` an einen Befehl wird der Inhalt der Datei **meinedatei** als Eingabe verwandt. Die Datei **meinedatei** wird also für das Programm zum **stdin**. Durch Anhängen von `> meinedatei` wird die Ausgabe auf die Datei **meinedatei** umgeleitet. Also wird die Datei

meinedatei für das Programm zum **stdout**. Beides ist kombinierbar. Beispielsweise bedeutet

```
sort <eingabedatei >ausgabedatei
```

dass die Datei **eingabedatei** als Eingabe für den Sortierbefehl verwendet wird und dass die Ergebnisse in der Datei **ausgabedatei** abgelegt werden.

```
cat >testdatei
```

Mit diesem Kommando kann man auch dann Textdateien erzeugen, wenn kein Editor verfügbar ist. Hier werden zwei Effekte ausgenutzt. Der erste ist die Umleitung der Ausgabe. Die Ausgabe von cat wird nicht am Bildschirm angezeigt, sondern in eine Datei geschrieben. Es wird also die Datei **testdatei** erzeugt. Der zweite Effekt ist, dass keine Eingabedatei für cat angegeben wird. Dementsprechend greift cat auf die Standard-eingabe, nämlich die von der Tastatur zu. Es wird also so lange von der Tastatur eingelesen, bis diese Eingabeeinheit am Ende der Datei ist. Das Ende der Datei wird bei der Tastatur mit **ctrl-D** erzeugt. Das Erzeugen von Dateien auf diese Weise ist natürlich hochgradig unbequem, da es kaum Korrekturmöglichkeiten gibt.

Editor für Arme

Beim Umleiten einer Ausgabe wird die Zieldatei zunächst geleert. Diesen Effekt kann man nutzen, wenn eine Protokolldatei zu groß wird. Da solche Dateien von anderen Prozessen beschrieben werden, kann man sie nicht einfach löschen. Auch wenn man die Datei unter diesem Namen wieder erzeugt, ist es nicht dieselbe Datei, die der Hintergrundprozess bearbeitet hatte. Mit dem Größerzeichen und dem Dateinamen wird die Datei sofort auf 0 Byte zurückgesetzt. Dabei ist die Datei nicht gelöscht worden. Es ist exakt die Datei, auf die der Hintergrundprozess im Zugriff hatte.

Datei stutzen

```
> /var/log/messages
```

Nicht immer soll der Inhalt der Datei gelöscht werden, in die man die Ausgabe umleitet. Verwendet man statt eines Größerzeichens zwei, so wird die Ausgabe an die existierende Datei angehängt.

>> hängt die
Ausgabe an die
Datei an

Um **stderr** umzuleiten, wird eine 2 vor das Größerzeichen geschrieben. Dies ist beispielsweise wichtig, wenn man die Fehlermeldungen eines Compilers in einer Datei auffangen will.²³

2> leitet stderr um

²³ Das funktioniert nicht in der C-Shell.

```
cc mistprogramm.c 2>fehlerliste
```

Um **stdout** und **stderr** in die gleiche Datei umzuleiten, gibt es je nach verwendeter Shell zwei Umleitungsoperatoren. Auf die verschiedenen Shells wird ab S. 98 eingegangen. Bei der Korn-Shell wird an das Größerzeichen ein kaufmännisches Und mit einer 1 angehängt. Bei der C-Shell wird nur das kaufmännische Und vor das Größerzeichen gestellt. Soll der Compiler des obigen Beispiels Fehlermeldungen und Ausgaben in dieselbe Datei umleiten, lauten die alternativen Befehle:

```
cc mistprogramm.c 2>&1 fehlerliste
cc mistprogramm.c &> fehlerliste
```

**/dev/null ist der
Müllschlucken**

Manchmal kann man mit den Ausgaben der Programme überhaupt nichts anfangen. Für solche Zwecke hat UNIX ein Datengrab oder einen Mülleimer, der **/dev/null** heißt. Alle Daten, die auf diese Pseudodatei umgeleitet werden, verschwinden auf Nimmerwiedersehen.

1.10.3 Piping

**Seitenweises
Blättern**

Durch den senkrechten Strich **|** wird eine Pipe aufgebaut, die die Ausgabe des links stehenden Kommandos auf die Eingabe des rechts stehenden Kommandos umleitet. Programme, die typischerweise im Datenstrom einer Pipe verwendet werden, nennt man Filter. Ein ganz typischer Filter ist das Programm **more**. Um sich längere Verzeichnisse seitenweise anzusehen, wird man eine Pipe zwischen **ls -l** und **more** verwenden:

```
ls -l | more
```

Nach jeder Bildschirmseite erscheint die Meldung **more**. Mit der Leertaste kann eine Seite weitergeblättert werden. Die Return Taste bewirkt ein zeilenweises Blättern. Will man die Ausgabe beenden, gibt man ein **q** ein.

**Bindestrich als
Dateiname**

Einige Programme geben ihre Ergebnisse nicht auf **stdout** aus, sondern benutzen immer eine Ausgabedatei. Dazu gehört beispielsweise das Datensicherungsprogramm **tar** (siehe S. 192). **tar** schreibt normalerweise auf das Standardgerät zur Datensicherung. Auf Servern ist dies meist die Bandstation. Allerdings kann man **tar** mit der Option **-f** dazu bringen, in eine Datei zu schreiben. In bestimmten Situationen soll das Programm aber in eine Pipe schreiben. Damit in solchen Fällen in eine Pipe hineingeschrieben werden kann, verlangen die Programme einen Dateinamen für die Pipe. An dieser Stelle wird ein Bindestrich als Synonym für **stdin** und **stdout** verwendet. Diese Konstruktion ist beim Kopieren von Verzeich-

nisbäumen mit `tar` (siehe S. 88) und beim Brennen von Verzeichnissen auf CD (siehe S. 120) zu sehen.

Datenabzweigung: `tee`

Manchmal kann man in einer Pipe auch die Zwischenergebnisse gut gebrauchen. In diesem Fall kann man das Kommando `tee`, gefolgt von einem Dateinamen, in die Pipe einfügen. Dann wird der aktuelle Datenstrom in diese Datei kopiert, während trotzdem die nächste Anwendung in der Pipe den Datenstrom unverändert weiterverarbeiten kann. Beispiel:

```
ls | tee out | wc
```

In der Datei **out** wird man die Dateinamen des aktuellen Verzeichnisses finden, während die Ausgabe des Befehls die Anzahl der Dateien darstellt. Das Programm `tee` bildet also das T-Stück im Datenstrom und zweigt Daten in die angegebene Datei ab.

1.10.4 Verschachtelte Befehlsargumente

Die Argumente eines Befehls können durch andere Prozesse erzeugt werden. Um dies zu erreichen, wird statt des Arguments ein Befehl in rückwärtige Hochkommata (backquotes) gesetzt.²⁴ Auf amerikanischen Tastaturen liegt diese Taste rechts neben dem Apostroph. Man findet die Taste auf einer deutschen Tastatur rechts neben dem ß in Kombination mit der Hochstelltaste. Das Ergebnis dieses Befehls wird dem eigentlichen Befehl als Argument übergeben. Beispiel:

```
ls -l 'cat filelist'
```

In diesem Fall wird der Inhalt der Datei **filelist** mit `cat` ausgegeben und so als Argumentliste dem `ls` übergeben.

Man könnte die Datei `filelist` als Liste der Dateien zu einem Programmierprojekt benutzen. Um diese zu übersetzen, könnte der Befehl

```
cc -o projekt 'cat filelist'
```

abgesetzt werden. Eine Sicherung der Quelltexte könnte dann per

```
cp 'cat filelist' /usr/projekt/backup
```

erfolgen. Leider sind die Backquotes im Ausdruck kaum von einem Apostroph zu unterscheiden. Beide haben aber unter UNIX eine deutlich

Befehl im Befehl

Alternativ-
schreibweise

²⁴ Von manchen UNIX-Anwendern wird dieses Zeichen auch »Rückwärtsdübel« genannt. Leider kann ich für die korrekte Schreibweise nicht garantieren.

unterschiedliche Bedeutung. Als Alternativschreibweise bieten die meisten Shells an, den in Backquotes stehenden Ausdruck einzuklammern und ein Dollarzeichen voranzustellen. Damit sind die beiden untenstehenden Ausdrücke gleichbedeutend:

```
ls -l 'cat filelist'
ls -l $(cat filelist)
```

1.11 Praktische Helfer

UNIX liefert eine Menge Werkzeuge, um mit Textdateien umzugehen. Man kann mit Bordmitteln fast beliebige Informationen aus Textdateien gewinnen und in beinahe unbegrenzter Form wieder zusammensetzen.

Viele Werkzeuge
für Textdateien

UNIX arbeitet sehr intensiv mit Textdateien, gerade im Bereich der Konfiguration und Administration. Diese an sich einfache Methode ermöglicht das leichte Auffinden von Fehlern. Es hat aber auch ungeheure Vorteile, wenn es darum geht, eine Installation zu dokumentieren, zu sichern und zu prüfen. Man kann Textdateien leicht ausdrucken oder kopieren. Manipulationen am System sind schnell erkannt und können so leicht beseitigt werden.

1.11.1 Ausgabe einer Datei: cat

Der Befehl `cat` zeigt den Inhalt von Dateien auf dem Bildschirm an, die ihm als Argumente angegeben werden. `cat` ist die Abkürzung für concatenate, also aneinander hängen. Durch eine Umleitung der Ausgaben (siehe S. 72) kann aus mehreren Dateien eine zusammenhängende Datei gemacht werden. Man kann dies auf dem Bildschirm sehen, wenn man mehrere Dateien hinter dem Befehl `cat` angibt. Folgender Befehl zeigt zweimal `hugo` und einmal `erna` direkt hintereinander an.

```
cat hugo hugo erna
```

1.11.2 Seitenweise: more

Um eine Datei anzuzeigen, deren Inhalt größer als der Bildschirm ist, verwendet man das Programm `more`. Es zeigt die Datei an und stoppt die Ausgabe am unteren Bildschirmrand. Mit der Leertaste kann man seitenweise, mit der Return-Taste zeilenweise weiterblättern. Mit `q` kann man die Ansicht vorzeitig verlassen.

less entspricht
more

Auf manchen Systemen wird das Programm `less` eingesetzt. Der Name stammt aus der Wortspielerei »less is more than more«. Mit `less` kann

man beispielsweise mit dem Buchstaben b auch rückwärts blättern. Auf vielen Systemen funktionieren sogar die Cursortasten zum Bewegen innerhalb des Textes.

Neben der einfachen Blätterfunktion ist es auch möglich, Textstellen zu suchen. Wie beim vi verwendet man den Schrägstrich. Der Cursor geht hier wie dort in die unterste Zeile und erwartet einen Suchbegriff. Allerdings verarbeitet more keine regulären Ausdrücke (siehe S. 84). Die nächste Fundstelle erreicht man wie bei vi mit einem n.

Suche nach
Stichworten

1.11.3 Durchsuchungsbefehl: grep

Der Befehl grep dient dazu, aus einer oder mehreren Dateien eine bestimmte Zeichenkette herauszusuchen und anzuzeigen. Der erste Parameter von grep ist das Suchmuster. Es folgen die zu durchsuchenden Dateien. Wurden mehrere Dateien angegeben, wird bei der Ausgabe der Ergebnisse auch der Dateiname ausgegeben.

Beispielsweise kann man in einem Projekt in C ermitteln, welche Module eine bestimmte Funktion (als Beispiel printf) verwenden. Zu diesem Zweck gibt man Folgendes ein:

```
grep printf *.c
```

Suchbegriffe mit Leerzeichen setzt man in Anführungszeichen oder Hochkommata.

Optionen

Der Befehl grep hat einige wichtige Optionen. Mit der Option -h (h wie hide; engl. verstecken) wird die Anzeige der Dateinamen unterdrückt. Das ist wichtig, wenn die ausgefilterten Ausgaben von anderen Programmen weiterverarbeitet werden sollen.

-h unterdrückt die
Dateianzeige

Die Option -v invertiert die Suche. Das heißt, es werden die Zeilen angezeigt, in denen das Muster eben nicht enthalten ist.

-v invertiert die
Ausgabe

Die Option -w legt fest, dass das Suchmuster ein vollständiges Wort darstellt. Worte werden durch Leerräume und Sonderzeichen begrenzt.

-w erfasst nur
Worte

Mit der Option -i kann die Beachtung von Groß- und Kleinschreibung aufgehoben werden. Die Suche nach einer hose findet dann auch Hose, HOSE und hose.

-i ignoriert groß
und klein

-c zählt nur Mit der Option -c (c wie count; engl. zählen) werden die Zeilen gar nicht ausgegeben, sondern es wird nur ermittelt, wie oft das Suchmuster in den Dateien vorkommt.

1.11.4 Wenn ich auf das Ende sehe: tail

Mit dem Kommando tail kann man sich die letzten Zeilen einer Datei anzeigen lassen. Als Argument wird die auszugebende Datei angegeben. Wieviel angezeigt werden soll, kann durch ein Minus, gefolgt von der Zahl der Zeilen, angegeben werden. Standard sind 10 Zeilen.

tail -f verfolgt Datei-veränderungen Ein wichtiges Einsatzgebiet von tail ist das Beobachten sich verändernder Dateien, wie beispielsweise Logfiles. Dazu dient die Option -f. Will man die messages-Datei beobachten, gibt man beispielsweise

```
tail -f /var/log/messages
```

ein. Alle Daten, die in diese Datei fließen, erscheinen nun auch auf dem Terminal. Durch die Delete-Taste oder ctrl-C (je nachdem, was auf dem lokalen System als Unterbrechungstaste definiert ist) kann der tail-Befehl wieder gestoppt werden.

Analog zu tail gibt es auch ein Programm head, das die ersten Zeilen einer Datei anzeigt. Seine praktische Bedeutung ist aber geringer, weil es sehr viel häufiger vorkommt, dass man die letzten Einträge einer Protokolldatei sehen will als die ersten.

1.11.5 Wortzähler: wc

Auch Zeilen und Zeichen zählen Dieses Programm mit dem eingängigen Namen (wc steht allerdings eigentlich für wordcount) zählt die Wörter in einer Datei. Wer nicht weiß, welchen Nutzen es ihm bringen soll, wenn er die Anzahl der Wörter in seinem C-Programm ermitteln kann, wird vielleicht eher eine Anwendung finden, wenn er weiß, dass das Programm mit der Option -l auch Zeilen und mit der Option -c Buchstaben zählt.

1.11.6 sort

Mit dem Befehl sort wird eine Datei sortiert auf dem Bildschirm ausgegeben. Wird kein weiterer Parameter angegeben, sortiert sort die Datei Zeile für Zeile nach ihren Anfängen in alphabetischer Reihenfolge. Besonders interessant ist dieses Programm natürlich dadurch, dass man die Ausgabe wieder in eine Datei umleiten und weiterverarbeiten kann.

Die wichtigsten Optionen sind `-r` für das Umkehren der Reihenfolge, `-f` für das Ignorieren der Klein- und Großschreibung und `-n` für die numerische Sortierung von Zahlen.

1.11.7 sed

`sed` ist das Kürzel für stream editor. Man könnte das mit »Datenstromtextverarbeitung« übersetzen. Das Programm wird hauptsächlich im Zusammenhang mit Pipes als Filter eingesetzt. Fast immer wird die Option `-e` angegeben, die Befehle auf den Datenstrom anwendet. Ein Befehl von `sed` hat folgenden Aufbau: Zunächst wird der Bereich genannt, in dem der Befehl wirkt. Es folgt der Befehlsname, und schließlich folgen, falls erforderlich, die Argumente des Befehls.

sed wird in Pipes eingesetzt

Der Wirkungsbereich wird wie bei `vi` in der Befehlszeile beschrieben. Der Aufbau einer Bereichsangabe ist: Anfangszeile, dann ein Komma, gefolgt von der Endzeile. Dabei steht das Dollarzeichen für die letzte Zeile. Auch der Befehl zum Suchen und Ersetzen, der im `vi` schon so viel Bewunderung auslöste, kann hier wiederverwendet werden. Weil es so schön war, nochmal das Beispiel:

Suchen und Ersetzen

```
1,$ s/suchmuster/ersetze/g
```

Beispielsweise kann man in einem Text jedes Auftreten von »cool« durch »erhebend« ersetzen:

```
cat kids.text | sed -e "1,$ s/cool/erhebend/g" > alten.tauglich
```

Man kann Zeilen aus dem Datenstrom mit dem Befehl `d` herauslöschen. Im Beispiel werden alle Zeilen entfernt, in denen das Stichwort »unfug« auftritt.

Löschen

```
cat normal.text | sed -e "/unfug/d" > kluger.text
```

1.11.8 awk

Das Programm `awk` ist ein Werkzeug zur Auswertung von Textdateien, die wie Datenbanktabellen behandelt werden. Der Name des Programms setzt sich aus den Anfangsbuchstaben der Nachnamen seiner prominenten Entwickler Aho, Weinberger und Kernighan zusammen. Das Programm `awk` ist in der Lage, aus einer Datei Spalten und Zeilen auszuwählen und darüber Auswertungen durchzuführen. Im Grunde ist es ein `grep`, das um die Fähigkeit erweitert wurde, auf Texte innerhalb der gefundenen Zeilen zuzugreifen, sie in Variablen aufzunehmen und zu verarbeiten.

awk kann Texte auswerten

awk besitzt darüber hinaus eine kleine Programmiersprache, sodass auch Fallunterscheidungen oder Ähnliches leicht zu realisieren sind.

Wie das aber bei flexiblen und leistungsstarken Programmen so ist, fühlt man sich auch leicht von der Komplexität der Möglichkeiten erschlagen. Dennoch lohnt sich ein Blick auf dieses Programm, da es bestimmte Probleme mit erstaunlich wenig Aufwand lösen kann.

Spalten durch Leerzeichen und Tabulatoren

awk betrachtet wie gesagt eine Textdatei als Datenbanktabelle aus Zeilen und Spalten. Die Zeilen sind durch den Zeilentrenner bestimmt, der unter UNIX übrigens mit dem Linefeed codiert wird (ASCII-Zeichen 10). Zwei Spalten werden aus der Sicht von awk und auch anderer UNIX-Werkzeuge durch ein oder mehrere Leerzeichen oder Tabulatorzeichen getrennt.

Suchen wie mit sed

awk kann wie grep anhand eines Suchbegriffes Zeilen aus einer Datei heraussuchen. Dazu wird der Suchbegriff in Schrägstriche eingeschlossen, was wiederum vom sed bekannt vorkommt. In geschweiften Klammern können Spalten ausgegeben werden. Der gesamte Befehl wird wiederum in Hochkommata eingeklammert, damit die Shell nicht daran herumwerkelt. Beispiel:

```
awk '/Willemer/ {print $3}' telefon
```

Dieser Befehl gibt für alle Zeilen der Datei **telefon**, in denen die Zeichenfolge Willemer steht, die dritte Spalte aus. Die Datei **telefon** ist eine einfache Textdatei, in der Name, Vorname und Telefonnummer in jeder Zeile stehen. Im Folgenden sehen Sie einen kurzen Ausschnitt:

```
Müller Anton 0987-6543  
Schmidt Erwin 01234-5678  
Willemer Arnold 04632-110
```

Das Hochkomma

Im Allgemeinen bestehen die Argumente von awk aus mehr als einem Wort. Da die Shell dem Programm die durch Leerzeichen getrennten Eingaben einzeln übergibt, wird ein Argument, das aus mehreren Wörtern besteht, in Anführungszeichen oder in Hochkommata eingeschlossen. Diese sind auch hilfreich, wenn das Kommando über mehrere Zeilen geht, da die Shell den Befehl erst als abgeschlossen ansieht, wenn das zweite Hochkomma erscheint. Man verwendet übrigens bei awk das Hochkomma statt des Anführungszeichens, da die Shell in Anführungszeichen auch Umgebungsvariablen auswertet. Das wird durch Hochkommata ausgeschlossen.

Selektion: Auswahl von Zeilen

Das Kommando zur Selektion wird gegeben, indem der Suchbegriff in Schrägstriche gesetzt wird. Unter einer Selektion versteht man die Auswahl der Zeilen einer Tabelle.

Suchbegriff
zwischen /
selektiert Zeilen

```
awk /Willemer/ telefon
```

Alle Zeilen mit der Zeichenfolge Willemer in der Datei **telefon** werden ausgegeben. Wird kein Suchbegriff angegeben, werden alle Zeilen angezeigt. Im Suchbegriff können reguläre Ausdrücke verwendet werden (siehe S. 84).

Würde die Datei **telefon** alle Telefonnummern enthalten, würde nach der Ausführung des Befehls meine gesamte Verwandtschaft einträchtig beisammen stehen, wie das im Leben vielleicht nicht der Fall wäre. Sind Computer nicht etwas Wunderbares?

Projektion: Auswahl von Spalten

Zur Projektion muss die Spalte angegeben werden und ein Kommando, was mit ihr passieren soll. Als Spaltentrenner gilt jede beliebige Kombination von Tabulatorzeichen oder Leerzeichen. Das einfachste Kommando ist die Ausgabe mittels `print`. Die Spalte wird durch deren Nummer und ein vorgestelltes Dollarzeichen bezeichnet. Die Zählung beginnt bei 1, da \$0 die gesamte Zeile bezeichnet. Aktionen werden immer in geschweifte Klammern gesetzt. Die dritte Spalte einer Datei wird also folgendermaßen ausgegeben:

Geschweifte
Klammern

```
awk '{print $3}'
```

Es ist möglich, mehrere Befehle in einer geschweiften Klammer zu setzen. Dabei sind die Kommandos jeweils durch ein Semikolon zu trennen.

Kombination von Kommandos

Die Kombination beider Optionen ist möglich.

```
awk '/Willemer/ {print $3}' telefon
```

Das bereits bekannte Beispiel selektiert also nach der Zeichenfolge Willemer und führt in allen Zeilen, auf die die Selektion zutrifft, den Inhalt der geschweiften Klammer aus. Dieser besagt, dass die dritte Spalte ausgegeben werden soll.

Variablen

Innerhalb des awk-Kommandos können Variablen verwendet werden. Variablen sind dem Programmierer aus den Programmiersprachen vertraut. Es sind einfach Speicherplätze für Informationen wie Zahlen oder Zeichenfolgen. Um auf diese Speicher zugreifen zu können, verwendet man Namen. Solche Variablen kennt auch awk, und sie können beispielsweise zur Bildung von Summen eingesetzt werden. Eine Deklaration, wie in vielen Programmiersprachen, ist nicht erforderlich.

Auch die Shell kennt Variablen, wie später näher erläutert wird. Im Gegensatz zu den Umgebungsvariablen der Shell wird den Variablen von awk kein Dollarzeichen zur Auswertung vorangestellt. Im folgenden Beispiel ist `anzahl` eine Variable, die dort zum Zählen der Tage verwendet wird.

Start-, Standard- und Endanweisungen

Man kann drei Typen von Anweisungen unterscheiden. Die Standardanweisung wurde bereits vorgestellt. Dieser Typ wird auf jede selektierte Zeile angewandt. Daneben gibt es noch die Startanweisung, die genau einmal vor der ersten Verarbeitung ausgeführt wird und der man das Schlüsselwort `BEGIN` voranstellt. Die Endanweisung wird durch das Schlüsselwort `END` eingeleitet. Hier ein Beispiel:

```
gaston> awk 'BEGIN{print"Wir zählen Tage!"; anzahl=0}
             {print $1;anzahl++}
             END{print "Das waren " anzahl " Tage"}}' tage
Wir zählen Tage!
montag
dienstag
mittwoch
donnerstag
freitag
samstag
sonntag
Das waren 7 Tage
gaston>
```

Die Datei **tage** enthält in der ersten Spalte die Wochentage. Bei `BEGIN` wird die lokale Variable `anzahl` auf null gesetzt. Das ist allerdings an sich nicht nötig, da Variablen bei awk standardmäßig mit null vorbesetzt sind. In der Standardanweisung wird nicht nur die erste Spalte ausgegeben, sondern parallel dazu die Variable `anzahl` um eins erhöht. Die `END`-Anweisung wird benutzt, um die Anzahl der Tage auszugeben.

Ein weiteres Beispiel soll die Summe aller Dateilängen mit der Endung **gnt** bilden. In diesem Fall wird **awk** an eine Pipe gehängt. Man gibt in der ersten Zeile das Kommando, um ein Verzeichnis in seiner Langform anzusehen. Das Ergebnis wird **awk** per Pipe zugeschoben. In der Standardanweisung von **awk** wird eine Variable um den Wert in der fünften Spalte erhöht. Die Endanweisung gibt diese Variable aus.

```
ls -l *.gnt | awk '{sum+=$5}END{print sum}'
```

Die Suche nach **gnt** könnte natürlich auch von **awk** übernommen werden. Dann hieße der Befehl:

```
ls -l | awk '/gnt/ {sum+=$5} END{print sum}'
```

Der Unterschied besteht darin, dass in diesem Fall Zeilen, in denen irgendwo **gnt** (beispielsweise im Gruppennamen) vorkommt, auch mitgerechnet werden.

Eingebaute Funktionen

awk hat einige eingebaute Funktionen, deren Syntax sehr an C erinnert.

Funktion	Wirkung
print	Anzeigen
printf	Entspricht der C-Funktion printf
sprintf	für die Speicherung von Ausgaben in einer Variablen
length(s)	ermittelt die Länge einer Zeichenkette
substr(s, anf, anz)	Teilstringbildung
index(s, t)	gibt die Stelle an, an der in s die Zeichenkette t beginnt
sqrt(n)	Wurzel
log(n)	natürlicher Logarithmus
exp(n)	e^n , $e=2,71828$
int(n)	ganzzahliger Anteil

Tabelle 1.23 awk-Funktionen

Ausführung von Programmdateien

Mit den Variablen und Befehlen kann man komplexe Programme erzeugen. Damit diese nicht jedes Mal neu eingegeben werden müssen, kann man Dateien mit den auszuführenden Aktionen schreiben. Der Dateiname wird **awk** mit der Option **-f** als Parameter übergeben.

```
awk -f awkprg
```

In diesem Beispiel stehen in der Datei **awkprg** die Befehle, die durch awk abgearbeitet werden sollen.

1.11.9 Weitere Werkzeuge im Überblick

Es gibt noch eine Reihe sehr praktischer kleiner Werkzeuge. Häufig sind es kleine Programme, die sehr spezielle Aufgaben lösen. Sie sind manchmal beinahe unbekannt, aber vielleicht in der einen oder anderen Situation hilfreich. Da sie zugegebenermaßen ein wenig exotisch sind, sollen sie hier nur tabellarisch aufgezählt werden.

Programm	Funktion
split	teilt eine große Textdatei in mehrere kleine auf
cut	schneidet einen angegebenen Teil aus einer Datei heraus
fold	umbricht Zeilen ab einer bestimmten Zeilenlänge
tr	wandelt Zeichen um
diff	zeigt die Unterschiede zwischen zwei Textdateien (siehe S. 553)

Tabelle 1.24 Diverse UNIX-Werkzeuge

Genauere Informationen finden Sie in den passenden Manpages.

1.12 Reguläre Ausdrücke

Für den Suchbegriff kann in vielen UNIX-Programmen wie grep oder vi ein regulärer Ausdruck verwendet werden. Zunächst einmal ist ein regulärer Ausdruck nichts anderes als ein Suchbegriff, und man kann ganz naiv den Begriff verwenden, den man sucht. Wenn Sie also das Wort »Maus« suchen, können Sie auch »Maus« als regulären Ausdruck angeben. Reguläre Ausdrücke können komplexeste Suchmuster beschreiben. Dann sehen diese Ausdrücke auf den ersten Blick allerdings auch ziemlich erschreckend aus.

Anders als Wildcards

Zunächst werden einfache Platzhalter verwendet. Bei den Dateimasken der Shell, den so genannten Wildcards, gibt es solche Platzhalter auch. Die regulären Ausdrücke haben allerdings nichts mit den Wildcards zu tun, die die Shell verwendet. Dort hat der Stern beispielsweise eine andere Bedeutung als hier. Das einfachste Sonderzeichen ist der Punkt. Er steht stellvertretend für genau ein beliebiges Zeichen. Die Suche nach M..s findet die Worte Maus, Moos und Muks, aber nicht Murks, da hier

zwischen M und s drei Zeichen stehen. Der Punkt ist also in der Wirkung mit dem Fragezeichen bei den Wildcards vergleichbar.

Der Stern und das Pluszeichen sind Multiplikatoren und beziehen sich immer auf das Zeichen links neben sich. Das Pluszeichen besagt, dass das Zeichen einmal oder mehrfach auftreten kann. Beim Stern ist es auch denkbar, dass das Zeichen gar nicht erscheint. Die Suche nach `abc*` findet also `abc`, `abcc`, `abcccccc`, aber auch `ab`. Wirklich interessant werden beide Zeichen in Verbindung mit dem Punkt. So findet `M.*s` `Maus` und `Moos`, aber eben auch `Murks` und `Meeresfrüchte`.

Multiplikatoren

Hier werden Sie vielleicht stutzen, denn `Meeresfrüchte` enden doch gar nicht auf `s`. Das ist richtig, aber im regulären Ausdruck wurde ja auch gar nicht erwähnt, dass das Wort hinter `s` enden soll. Das müsste man explizit mit einem `\>` angeben. Das Gegenstück lautet `\<` und bedeutet Wortanfang. So wie nach dem Wortanfang und -ende gesucht werden kann, so gibt es auch das `^` für den Zeilenanfang und das `$` für das Zeilenende. Eine wichtige Anwendung ist, die Verzeichnisse anzeigen zu lassen. Unter UNIX unterscheidet man Dateien von Verzeichnissen an dem kleinen `d` am Zeilenanfang, wenn man `ls -l` aufruft. Dementsprechend würde folgende Befehlskombination nur die Verzeichnisse anzeigen:

Anfang und Ende

```
gaston> ls -l | grep ^d
drwxr-xr-x    3 arnold   users          4096 Jun 25 20:57 pic
drwxr-xr-x    2 arnold   users          4096 Jun 28 20:55 unprog
gaston>
```

`grep` sucht in der Standardeingabe ein `d`, das direkt dem Zeilenanfang folgt, oder anders ausgedrückt, das am Anfang der Zeile steht. Ohne das Dach hätte man alle Zeilen erhalten, in denen ein `d` steht. Da der Benutzer `arnold` heißt, wären das wohl alle Dateien des Verzeichnisses.

Vielfältige Möglichkeiten gewinnt man im `vi` dadurch, dass man als Suchwort einen regulären Ausdruck verwenden kann. Ganz besondere Möglichkeiten tun sich dadurch auf, dass man Markierungen innerhalb eines Ausdrucks setzen kann und diese beim Ersetzen verwenden kann. Ein praktisches Beispiel findet sich beim Umsetzen von $\text{T}_{\text{E}}\text{X}$ -Dokumenten nach HTML. In der ersten Zeile sehen Sie eine Überschrift in $\text{T}_{\text{E}}\text{X}$ und darunter eine in HTML:

Ersetzen im vi

```
\section{Dies ist ein spannendes Kapitel}  
<H1>Dies ist ein spannendes Kapitel</H1>
```

Um alle Vorkommen von `section` in die entsprechenden `<H1>` umzuwandeln, wird ein regulärer Ausdruck verwendet. Zunächst wird das Muster beschrieben, das eine `section` erkennt.

```
\\section{.*}
```

Der doppelte Backslash muss sein, damit er nicht fälschlich als Kommando interpretiert wird. In den geschweiften Klammern steht schlicht Punkt Stern, also der Ausdruck für eine beliebige Zeichenfolge. Das ist unsere Überschrift, die wir gern übernehmen wollen. Also wird davor und dahinter eine Markierung gesetzt.

```
\\section{\\(.*\\)}
```

Nun wird das Ganze in den Ersetzungsbefehl von `vi` eingesetzt. Der komplette Aufruf lautet also:

```
:1,$ s/\\section{\\(.*\\)}/<H1>\\1</H1>/g
```

Der letzte Backslash der Zeile muss sein, sonst glaubt `vi`, dass der Schrägstrich des `</H1>` der Befehl dafür wäre, dass der Ersetzungsbereich hier endet. Die Zeichenfolge `\\1` in der Ersetzung liefert den in der Markierung gemerkten Wert und befördert die Überschrift in die gewünschte, neue Umklammerung.

Machen Sie sich klar, dass Sie sich mit diesem zugegeben etwas kryptischen Befehl vielleicht stundenlange Arbeit ersparen, wenn Sie in einem langen Dokument die Überschriften austauschen müssen. Und überlegen Sie sich auch, ob Sie so etwas mit einem normalen Editor ohne reguläre Ausdrücke auch könnten.

Der Grund, dass so viele Programme mit regulären Ausdrücken umgehen können, ist, dass UNIX dem Programmierer die Suche nach regulären Ausdrücken aus einer Bibliothek anbietet (siehe S. 630).

Ausdruck	Bedeutung
.	(Punkt) Steht für ein einzelnes beliebiges Zeichen.
[afg]	Das Zeichen a, f oder g muss an dieser Stelle erscheinen.
[0-9]	Eine Ziffer muss an dieser Stelle stehen.
*	Das vorangehende Zeichen kommt beliebig oft vor.
+	Das vorangehende Zeichen kommt mindestens einmal vor.
^	Zeilenanfang
\$	Zeilenende
\<	Wortanfang
\>	Wortende
\	Das folgende Zeichen wird nicht als Metazeichen interpretiert.
\(\)	Markierung eines Bereichs
\1 \2 ...	Referenz auf die erste und zweite Markierung

Tabelle 1.25 Reguläre Ausdrücke (regular expressions)

1.13 Pack deine Sachen und geh...

Wenn man ein Rudel Dateien, das vielleicht auch noch in einem größeren Verzeichnis untergebracht ist, transportieren möchte, dann hat man zwei Probleme. Erstens sollte das Paket handlich sein und zweitens möglichst klein. Dafür gibt es unter UNIX je eine Lösung: tar und compress. Und natürlich kann man auch beide kombinieren.

1.13.1 Verschnüren: tar

Das Programm tar (tape archiver) kommt aus dem Bereich der Datensicherung. Aber es ist ungeheuer praktisch im Umgang mit Dateien. Mit tar kann man packen, auspacken und Pakete anschauen. Gesteuert wird die Funktion durch die erste Option: c (create) zum Erzeugen, x (extract) zum Auspacken und t zum Anschauen. Genau eine von diesen Optionen braucht tar, damit klar ist, was zu tun ist.

Pflichtoption
c, x oder t

Da die Daten in eine Datei gepackt werden sollen, braucht man die Option f (file) für Datei. Denn dadurch arbeitet dann tar nicht auf dem Standardbandlaufwerk, sondern auf der angegebenen Datei. An dieser Stelle wird deutlich, dass tar ursprünglich für Magnetbänder entwickelt wurde. Und zu guter Letzt ist die Option v (verbose) hilfreich. Dann erzählt tar, welche Dateien bearbeitet werden.

Dateien in ein Archiv packen

```
tar -cvf ../meins.tar .
```

Dieser Befehl erzeugt (c) eine Datei (f) namens **../meins.tar** und zeigt (v), welche Dateien er einpackt. In die Archivdatei kommen das aktuelle Verzeichnis (.) und alle Unterverzeichnisse hinein. Dass die Unterverzeichnisse mitkommen, ist bei tar Standard und muss nicht angegeben werden. Dass die Datei **meins.tar** im Verzeichnis vor dem aktuellen Verzeichnis liegt (..), hat seinen praktischen Grund darin, dass tar so nicht in Versuchung kommt, das gerade erzeugte Archiv selbst ins Archiv aufzunehmen. Man kann das Problem auch dadurch umgehen, dass man sein Paket im Verzeichnis **/tmp** packt. Dass die Datei die Endung **tar** hat, ist nicht zwingend, sondern dient nur Ihrer Information.

Archiv wieder auspacken Nun sei die Datei per E-Mail oder Diskette oder wie auch immer auf den Zielrechner gelangt. Zum Auspacken hat man sie zunächst in das Verzeichnis **/tmp** gelegt. Um die Dateien auszupacken, wechselt man per **cd** in das Zielverzeichnis und weist dort tar an, das Archiv zu entpacken.

```
tar -xvpf /tmp/meins.tar
```

Absolute Pfadnamen Beim Packen und Auspacken sollte man sehr genau darauf achten, ob die Pfadnamen der gesicherten Dateien absolut sind, also mit einem Schrägstrich beginnen. In diesem Fall hilft die Option **-A**, dieses abzustellen. Sicherheitshalber sollte man die Manpage zu Rate ziehen, denn einige Systeme verwenden **-A** zum Anhängen an existierende Archive.

Dateieigenschaften Die Option **p** bewirkt, dass die Dateien die gleichen Rechte wie vor dem Einpacken haben. Wenn root den Befehl tar benutzt, wird die Option standardmäßig eingeschaltet. Jeder andere Benutzer muss sie explizit nennen.

Verzeichnisbaumkopie mit tar

Eine völlig andere Anwendung ist das rekursive Kopieren mit tar. Hier werden die zu kopierenden Daten in eine Pipe geschoben (tar c) und von dort an anderer Stelle wieder ausgepackt. Diese Vorgehensweise hat zwei Vorteile. Einerseits kann man so leicht ganze Verzeichnisbäume kopieren, und zweitens bleiben alle Dateiattribute erhalten.

```
cd /home/ernst  
tar cf - . | ( cd /home/august ; tar xfp - )
```

Der Befehl kopiert unter Beibehaltung aller Dateiattribute alle Dateien unterhalb des Verzeichnisses **/home/ernst** in das Zielverzeichnis **/home/august**. Solche Befehlskombinationen sind auf den ersten Blick etwas

erschreckend. Aber hier wird die besondere Stärke von UNIX ausgenutzt, Programme miteinander zu kombinieren. Zunächst wechselt der Befehl `cd` in das Startverzeichnis **/home/ernst**. Das ist wichtig, weil so den Dateien in der Archivdatei kein Pfadname vorangestellt wird. Der `tar`-Befehl links von der Pipe (`|`) weist dann keine Überraschung auf. Durch die Kombination `f -` wird statt eines Speichermediums oder einer Datei in die Standardausgabe **stdout** und damit in die Pipe geschrieben. Rechts von der Pipe wird also ein auspackender `tar`-Befehl stehen. Den kann man dort auch leicht identifizieren. Er steht in einer Klammer mit dem Befehl `cd`, der zunächst in das Zielverzeichnis **/home/august** wechselt. Nur so gelangen die Daten überhaupt an einen anderen Ort. Dort holt `tar` auf der rechten Seite die Daten aus der Pipe und legt sie dort ab, wo er steht. Das `p` gewährleistet auch für normale Anwender, dass alle Rechte der Dateien so bleiben, wie sie vor dem Kopieren waren.

1.13.2 Zusammenpressen: `compress` und `gzip`

Auf jeder UNIX-Maschine findet man das Tool `compress` und sein Gegenstück `uncompress`. `compress` komprimiert die als Argument genannten Dateien und kennzeichnet Sie, indem es ein großes `.Z` an den Namen hängt. Durch die Komprimierung können Textdateien leicht auf ein Drittel ihrer Größe schrumpfen. Die Originaldateien werden nach erfolgreichem Komprimieren gelöscht. Die gepackte Datei kann mit `compress -d` oder mit dem Befehl `uncompress` wieder entpackt werden.

`compress` ist auf jeder UNIX-Maschine verfügbar

Inzwischen verfügt fast jede Maschine über `gzip` und `gunzip`. `gzip` (sprich: GNU zip) arbeitet wie `compress` dateienweise, hat aber einen etwas besseren Komprimierungsalgorithmus. `gunzip` kann auch von `compress` erzeugte Dateien entpacken. Umgekehrt funktioniert das aber nicht. Eine mit `gzip` gepackte Datei erkennt man an der Endung `.gz`. Ergänzt wird `gzip` durch weitere praktische Tools. `zless` arbeitet wie `more` (siehe S. 76), allerdings auf einer gepackten Datei. Sehr hilfreich ist auch `zgrep`, mit dem man mehrere gepackte Dateien nach Stichwörtern durchsuchen kann, ohne sie einzeln auspacken zu müssen. Diese Befehle sind besonders beim Arbeiten mit den HowTos²⁵ hilfreich, die standardmäßig als `gz`-Dateien vorliegen.

GNU zip enthält viele Tools

1.13.3 Kombination aus Packen und Pressen

Natürlich kann man unter UNIX leicht `tar` und `compress` kombinieren, indem man sie durch eine Pipe verbindet. In diesem Fall geht es aller-

²⁵ <http://www.linuxdoc.org>

dings einfacher. Da diese Anwendung so häufig vorkommt, wurde in die neueren Implementationen von tar eine Option z eingebaut, die das Packen automatisch einbindet. Je nach Version wird compress oder gzip als Komprimierer eingesetzt. Beispiel:

```
tar cvzf ../archiv.tgz .
```

Bevor man also Daten mit einem anderen System tauscht, sollte man prüfen, ob die Dateien auch kompatibel sind. Wird als Komprimierer gzip verwendet, verwendet man üblicherweise die Endung tgz.

ZIP-Tools Einige Maschinen besitzen auch zip und unzip, die kompatibel zu den ZIP-Tools unter MS-DOS oder MS Windows sind. Das bedeutet, dass sie die typischen ZIP-Dateien, die man im Internet findet, aus- und einpacken können. Im Unterschied zum oben genannten gzip enthalten diese Werkzeuge auch einen Mechanismus, um mehrere Dateien zu einer Datei zusammenfassen zu können. Da tar auf den oben genannten Systemen fast unbekannt ist, verwendet man dort jedes Mal ZIP-Dateien, wenn mehrere Dateien zu einer zusammengefasst werden sollen.

```
zip projekt.zip *.c *.h
```

Mit diesem Kommando werden alle Dateien des Verzeichnisses mit den Endungen .c und .h in der Datei **projekt.zip** zusammengefasst und komprimiert. Wenn die Datei vorher noch nicht existiert hat, wird sie neu angelegt.

```
unzip projekt.zip
```

Alle in der Datei **projekt.zip** zusammengefassten Dateien werden im aktuellen Verzeichnis ausgepackt. Da ZIP-Dateien nicht aus dem UNIX-Umfeld stammen, sind sie nicht in der Lage, alle Attribute einer Datei zu sichern.

1.14 Prozesse

UNIX beherrscht Multitasking (siehe Glossar S. 637) und stellt die Möglichkeit, mehrere Prozesse parallel zu starten, dem Anwender mit einfachen Befehlen zur Verfügung. Ein Prozess ist die Bezeichnung für ein gestartetes Programm.

1.14.1 Hintergrundbearbeitung und Reihenfolge

Wenn man & an ein Kommando anhängt, wartet die Shell nicht auf das Ende des Programms, sondern lässt es im Hintergrund arbeiten. Dabei wird an der Konsole noch die Prozess-ID angezeigt, unter der

der Prozess gestartet wurde. Im Beispiel unten wird das Programm xman im Hintergrund gestartet. In der eckigen Klammer wird angezeigt, dass es bereits das zweite Programm ist, das so gestartet wurde. Der neue Prozess erhält die Prozess-ID 3717. Wird das Hintergrundprogramm irgendwann beendet, erscheint auch darüber eine Meldung auf der Konsole. Wieder wird die Jobnummer angezeigt und mit »Done« (engl. getan) angezeigt, dass das Programm xman fertig ist.

```
gaston> xman &  
[2] 3717  
gaston>  
...  
gaston>  
[2]+ Done xman  
gaston>
```

Man kann den Prozess betrachten, indem man den Befehl jobs angibt. Leider wird dieser Befehl von der Bourne-Shell nicht unterstützt. Aber der Befehl ps ohne Optionen zeigt die Liste der gestarteten Prozesse.

**Prozess-
beobachter**

Wenn man mehrere Befehle hintereinander in einer Kommandozeile angeben will, verwendet man als Trennzeichen das Semikolon. Die folgende Eingabe zeigt also erst die Dateien an und dann das Datum:

**Zwei Prozesse
nacheinander**

```
ls; date
```

Durch Anhängen eines & werden die beiden Prozesse nacheinander im Hintergrund ausgeführt. Das zweite Programm wartet also mit seinem Start, bis das erste Programm fertig ist. Sollen die Programme parallel laufen, schickt man beide einzeln in den Hintergrund:

```
gaston> ls &  
gaston> date &
```

Beim Starten der Prozesse im Hintergrund kann man beobachten, dass beide Programme ihre Ausgabe direkt auf den Bildschirm schreiben, von dem sie gestartet wurden. Da man im Allgemeinen Prozesse im Hintergrund nicht am Bildschirm beobachten will, sollte man die Ausgabe in eine Datei umleiten.

Terminalausgaben

Sollen zwei Programme nacheinander ausgeführt werden, tut man dies normalerweise, weil ein Programm auf den Ergebnissen eines anderen aufbaut. In solchen Fällen ist es aber sinnlos, das zweite Programm auszuführen, wenn bereits das erste scheiterte. Auch hierfür gibt es eine Lösung. Programme unter UNIX geben ihren Erfolgsstatus an die Shell in

**Voneinander
abhängige
Prozesse**

Form einer Fehlernummer zurück. Ist diese 0, ist alles in Ordnung. Durch ein doppeltes kaufmännisches Und, also &&, wird das zweite Programm nur gestartet, wenn das erste Programm 0 zurückgab:

```
programm1 && programm2
```

Auch das Gegenteil ist möglich: Durch die Verwendung zweier senkrechter Striche wird programm2 nur ausgeführt, wenn programm1 scheiterte:

```
programm1 || programm2
```

Klammern
bewirken eine
eigene Subshell

Durch Klammern eingeschlossene Kommandos werden in einer eigenen Subshell gestartet. Auf diese Weise lassen sich Kommandos bündeln. Da die Shell die Rückmeldung der zuletzt gestarteten Programme an den Aufrufer weiterleitet, kann man auch in den logischen Abfolgen klammern.

Tabelle 1.26 zeigt, wie mehrere Programme gestartet werden können.

Syntax	Ausführung
prog1 ; prog2	Erst prog1, dann prog2 ausführen
prog1 && prog2	prog2 nur bei Erfolg von prog1 ausführen
prog1 prog2	prog2 nur bei Misserfolg von prog1 ausführen

Tabelle 1.26 Start mehrerer Programme

1.14.2 Prioritäten: nice

Priorität
reduzieren

Der Befehl nice kann einen Prozess auf eine niedrigere Priorität setzen, damit andere Prozesse in ihrer Ausführung nicht so stark gestört werden. Spötter behaupten, es sei der unter UNIX am seltensten benutzte Befehl. Dabei ist die Benutzung extrem einfach: Vor das Kommando schreibt man einfach das Wort nice. Man kann den Grad seiner Nettigkeit sogar beziffern; sie kann maximal bei 19 liegen. Je höher der Wert, desto netter ist der Anwender, weil er die Priorität seines Prozesses zugunsten anderer Prozesse senkt. Gibt man seine Freundlichkeit nicht explizit an, ist sie 10. Nur root darf seine Nettigkeit in negativen Zahlen ausdrücken und darf dabei bis -20 gehen. Einige Systeme vermeiden die negativen Zahlen, indem sie den Wertebereich von 0 bis 39 festlegen.

Um einen Prozess im Nachhinein in seiner Priorität zu verändern, gibt es das Programm renice, das auf Seite 252 näher beschrieben ist. Das ist praktisch, wenn man einen Prozess in den Hintergrund gestellt hat, der so viel Aufmerksamkeit der Maschine erfordert, dass man nicht mehr im Vordergrund arbeiten kann.

1.14.3 Ausloggen bei laufendem Prozess: nohup

Wenn man einen Prozess mit & in den Hintergrund stellt und sich anschließend ausloggt, erhält dieser Prozess ein Signal namens SIGHUP (zu Signalen siehe S. 604). Dies führt, wenn der Programmierer es nicht explizit abfängt, normalerweise zum Abbruch des Programms. Um das Senden des Signals zu verhindern, gibt es den Befehl nohup, den man dem Befehl einfach voranstellt. Dadurch wird das Programm von SIGHUP nicht mehr belästigt, und man kann sich problemlos ausloggen.

Bei der Verwendung des Befehls nohup ist abzusehen, dass das Terminal für Ausgaben nicht zur Verfügung stehen wird. Die Ausgabe des Prozesses, die normalerweise auf das Terminal geht, wird in die Datei **nohup.out** umgeleitet. Falls der Prozess im aktuellen Pfad kein Schreibrecht hat, wird die Datei im Heimatverzeichnis abgelegt.

Automatische
Umleitung

1.14.4 Prozessliste anzeigen: ps

Der Befehl ps zeigt eine Prozessliste. Ohne Parameter zeigt er die Liste der eigenen Prozesse der aktuellen Sitzung. Das sind im Allgemeinen die Shell und ps selbst. Wurden in der Sitzung Prozesse in den Hintergrund gestellt, die noch laufen, erscheinen diese auch. Interessant an dieser Liste ist die PID. Das ist die Prozess-ID, die für jeden Prozess eindeutig ist. Verwendet man für ps den Parameter -l, dann sieht man diverse Informationen mehr über die Prozesse. Dabei gibt es eine PPID (Parent Process ID), die die PID des Elternprozesses ist. Der Vaterprozess ist derjenige Prozess, von dem ein Prozess gestartet wurde. Man kann erkennen, dass der Vater des ps die Shell ist. Es gibt einen Baum von Prozessen, der mit dem init-Prozess beginnt. Dieser hat immer die PID 1.

Auch die Prozess-
abhängigkeiten
bilden einen Baum

```
gaston> ps -l
UID    PID  PPID  PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
501   1292   1291   69    0 -   711 wait4  pts/0      00:00:00 bash
501   3578   1292   69    0 -   931 do_sel pts/0      00:00:03 vi
501   3598   3597   70    0 -   712 wait4  pts/2      00:00:00 bash
501   3634   3598   69    0 -  1984 do_pol pts/2      00:00:02 ggvs
501   3658   3634   69    0 -  2770 do_sel pts/2      00:00:03 gs
501   3807   3598   77    0 -   749 -      pts/2      00:00:00 ps
gaston>
```

In der Prozessliste ist zweimal die Shell bash zu erkennen, einmal mit der PID 1292, das andere Mal mit der PID 3598. Die erste Sitzung ist der Vater eines vi. Die zweite Sitzung ist einmal der Vater von ggvs, der selbst wieder

Vater von `gs` ist, und dann der Vater des `ps`. Man kann daraus schließen, dass `ggv` vor dem Aufruf des `ps` im Hintergrund gestartet wurde.

Man kann noch sehr viel mehr Details erkennen, wie die CPU-Zeit, die der Prozess bereits verbraucht hat, das Kommando, das ausgeführt wurde, von welchem Terminal der Prozess gestartet wurde und einige Dinge mehr.

Dies ist der Grund, warum man unter UNIX niemals ein Passwort oder etwas Geheimes als Kommando absetzen darf. Durch `ps` kann jeder andere Benutzer der Maschine den vollständigen Befehl lesen.

Fremde Prozesse Neben den eigenen Prozessen kann man auch die Prozesse anderer Benutzer sehen. Dazu gibt man `-alx` oder `-elf` an. Je nach System funktioniert der eine oder der andere Parameter. Danach rauschen alle Prozesse des Systems auf dem Bildschirms vorüber, was man durch eine Pipe nach `more` natürlich verhindern kann. Wichtig zu wissen ist, dass man auf einem UNIX-System jede Aktivität beobachten kann. Dadurch sind ein Fehlverhalten des Systems sowie Amok laufende Prozesse und Prozesse unbekannter Herkunft leicht identifizierbar.

Mit der Option `-u` wie User wird zusätzlich auch noch der Eigentümer des jeweiligen Prozesse angezeigt.

Die Option `-w` (w wie wrap) bricht die einzelnen Zeilen der Prozessliste am Zeilenende um. Das normale Verhalten von `ps` ist es, die Ausgabe am rechten Bildschirmrand abzuschneiden. So verliert man aber leicht die Informationen, die über den rechten Rand hinausragen.

1.14.5 Stoppen eines Prozesses: kill

Hat man einen Prozess in den Hintergrund gestellt, möchte ihn aber doch wieder vorzeitig stoppen, kann man dies mit dem Kommando `kill` erreichen. Als Parameter gibt man die Prozess-ID des zu terminierenden Prozesses an.

Lizenz zum Töten Man darf nur eigene Prozesse abschießen. Lediglich der Administrator ist berechtigt, auch fremde Prozesse zu terminieren. Die meisten UNIX-Programme sind so geschrieben, dass sie ihre Daten sichern und einen regulären Abgang durchführen, wenn sie ein Signal von einem normalen `kill` empfangen. Sollte ein `kill` also nicht sofort Erfolg zeigen, kann das damit zu tun haben, dass der Prozess noch mit den Terminierungstätigkeiten beschäftigt ist. Ein zweiter `kill` ist da selten hilfreich. Nach etwa fünf Sekunden sollten die Prozesse aber dann terminieren. So viel Zeit

gibt ihnen der Shutdown auch, bevor er weitergeht. Hartnäckige Prozesse schießt man mit einem `kill -9` ab. Zum Thema `kill` und Signale siehe S. 251.

1.14.6 Programmabbruch

Ein gestarteter Prozess kann meist mit `ctrl-C` abgebrochen werden. Auf älteren Systemen wurde dazu die Delete-Taste verwendet. Der im Vordergrund laufende Prozess erhält dadurch das Terminierungssignal `SIGINT` und beendet sich regulär.

Programmende
durch `ctrl-C`

Es gibt bei vielen UNIX-Systemen die Möglichkeit, einen Prozess auch kurzfristig anzuhalten. Allerdings muss auch die Shell dies unterstützen. Dazu drückt man `ctrl-Z` und erzeugt damit das Signal `SIGTSTP`. Es erscheint die Meldung, dass der Prozess gestoppt worden ist:

Programmunter-
brechung durch
`ctrl-Z`

```
[1]+  Stopped (user)          xemacs libash.htm
```

In der rechteckigen Klammer steht die Jobnummer aus Sicht der Shell, in diesem Fall eine 1. Diese Nummer darf nicht mit der PID verwechselt werden. Auf diese Nummer beziehen sich die Kommandos `fg`, `bg` und `kill`, wenn ein Prozentzeichen vorangestellt wird. Mit dem Befehl `jobs` wird die Liste der Jobs angezeigt. Man kann den Job im Vordergrund fortsetzen, indem man

```
fg %1
```

eingibt. Damit hat man die Situation vor dem `ctrl-Z` wieder hergestellt. Das kann Sinn machen, wenn man kurz das Terminal für andere Zwecke benötigte. Um den gestoppten Job im Hintergrund weiterlaufen zu lassen, gibt man den Befehl `bg` ein:

```
bg %1
```

Das macht vor allem dann Sinn, wenn man einen Prozess aus Versehen ohne `&` gestartet hatte und er nun die Konsole blockiert. Man sollte aber vorsichtig sein, die frei gewordene Konsole zum Ausloggen zu verwenden, da das `bg`-Kommando nicht nachträglich das `nohup`-Kommando setzt.

Letztendlich kann man den zunächst gestoppten Job auch terminieren:

```
kill %1
```

Insbesondere beim `kill` ist es wichtig, das `%`-Zeichen vor der Zahl nicht zu vergessen, da diese die Jobnummer und nicht die PID bezeichnet. Das Verfahren funktioniert nicht bei allen UNIX-Varianten, da das Signal `SIGSTP` benötigt wird, das nicht in jedem UNIX verfügbar ist. Auch die

Shell muss mitspielen. Die Korn-Shell und die bash von LINUX und Solaris 8 unterstützen es.

1.15 Umgebungsvariablen

UNIX stellt systemweite Variablen zur Verfügung. Jeder Prozess übergibt bei Start eines Kindprozesses auch die Umgebungsvariablen. Jeder Prozess kann den Inhalt der Variablen ändern oder neue hinzufügen. Allerdings können die Änderungen nur von den nachfolgenden Prozessen zur Kenntnis genommen werden. Die Shell bietet eine einfache Möglichkeit, Variablen zu setzen und abzufragen. In Shellskripten werden sie zum Festhalten von Ergebnissen verwendet. Aber nur die Variablen, die mit dem Kommando `export` bzw. bei der C-Shell `setenv` explizit in die Umgebungsvariablen gestellt werden, werden auch an die Kindprozesse weitergeleitet.

Man definiert eine Variable, indem man ihr einen Inhalt zuweist. Entsprechend wird sie wieder entfernt, wenn man ihr einen leeren Inhalt zuweist:²⁶

```
gaston> HUGO="schau an"
gaston>
```

Es darf kein Leerzeichen zwischen Variablenname, Gleichheitszeichen und Zuweisungsobjekt auftauchen. Die Variable HUGO wird angelegt, und ihr Wert ist ab diesem Augenblick die Zeichenkette »schau an«. Um den Inhalt einer Variablen auszulesen, stellt man ein Dollarzeichen voran:

```
gaston> echo $HUGO
schau an
gaston>
```

Variablen in Großbuchstaben

Der `echo`-Befehl führt zur Ausgabe der Wörter »schau an«. Die einschließenden Anführungszeichen werden nicht in der Variablen gespeichert. Übrigens müssen Variablennamen nicht zwingend in Großbuchstaben gesetzt sein. Da die vom System vorgegebenen Variablennamen groß sind, halten sich auch die meisten Anwender an diese Vorgabe. Aus Gründen der Übersichtlichkeit sollte man nicht ohne zwingenden Grund eine andere Konvention einführen.

Die Tabelle 1.27 zeigt einige vorbelegte Variablen.

²⁶ Im Folgenden wird die Syntax der Bourne-Shell und ihrer kompatiblen Nachfolger verwendet. Die C-Shell hat eine etwas andere Syntax.

Variable	Bedeutung
HOME	Heimatverzeichnis des Benutzers
PATH	Verzeichnisse, die nach Programmen durchsucht werden
PS1 PS2	Promptzeichen
IFS	Alle Zeichen, die wie ein Leerzeichen wirken sollen
PWD	Die Variable hält immer den Wert, den pwd liefert

Tabelle 1.27 Vorbelegte Environmentvariablen

Die Variable PATH enthält die Pfade, auf denen auf dieser Maschine die ausführbaren Programme zu finden sind. Die einzelnen Pfade sind durch einen Doppelpunkt voneinander getrennt. Die Variable muss nur selten geändert werden, da normalerweise die Startdateien für Programme an immer die gleichen Stellen installiert werden. Am interessantesten ist die Frage, ob der Punkt auch als Verzeichnis aufgelistet ist. Denn nur dann werden Programme aus dem aktuellen Verzeichnis auch gestartet, ohne einen Pfad anzugeben. In sicherheitskritischen Umgebungen wird der Punkt gern weggelassen, um es Trojanischen Pferden²⁷ schwerer zu machen. In solch einem Fall muss das Programm im aktuellen Arbeitsverzeichnis mit Namen **prog** durch `./prog` aufgerufen werden. Die Reihenfolge der Verzeichnisse in der Variablen PATH ist ebenfalls von Interesse, da in dieser Reihenfolge die Verzeichnisse durchsucht werden, ob das aufgerufene Programm dort zu finden ist. Wird eines gefunden, wird es gestartet und die Suche abgebrochen.

PATH

Da man bei Programmen, die im Pfad liegen, nur den Namen des Kommandos angibt, will man manchmal wissen, wo das Programm tatsächlich liegt bzw. welches Programm man ausführt, wenn mehrere Programme gleichen Namens in den Pfaden stehen könnten. Zu diesem Zweck gibt es die Programme `which` und `whereis`. Man gibt ihnen den Namen des Programms mit, und sie durchsuchen alle Pfade und geben das Programm mit vollständigem Pfad an.

**Pfadbestimmung
von Programmen**

PS1 ist der Prompt, also die Zeichenkette, die im Normalfall links neben dem Cursor steht. Standardmäßig steht dort für den normalen Anwender ein Dollarzeichen (\$) und für den Administrator ein Hashzeichen (#). In-

**PS1 bestimmt den
Prompt**

²⁷ Ein Trojanisches Pferd ist ein Programm, das unter dem Namen eines anderen Programms auf den Rechner geschmuggelt wird. Meist steckt die Absicht dahinter, Programme unter fremden Rechten ausführen zu lassen oder Passwörter auszu-spionieren.

zwischen ist es aber Mode geworden, dort alle möglichen Informationen abzulegen. Dadurch füllt auf manchem System der Prompt zwei Drittel der Zeile. Andererseits ist es sicher nicht verkehrt, wenn man darüber informiert wird, wer man eigentlich ist und wo man sich befindet. So bekommt man auf meiner Maschine auf den Befehl `echo $PS1` die Antwort:

```
\u@\h:\w >
```

Die Fähigkeit, den Anwender (`\u`), den lokalen Rechner (`\h`) und das aktuelle Verzeichnis (`\w`) im Prompt anzuzeigen, ist allerdings spezifisch von der Shell abhängig. So sieht der Prompt neben dem Cursor so aus:

```
arnold@gaston:~/my/texte/tex/buch >
```

- PS2** Die Variable `PS2` bestimmt den Sekundärprompt. Er wird erzeugt, wenn das angefangene Kommando nach dem Return noch nicht beendet ist. Dies merkt die Shell bei Anführungszeichen oder Klammern, die gesetzt wurden, aber deren Gegenstück nicht eingegeben wurden. `PS2` ist normalerweise ein einfaches Größerzeichen und wird auch selten verändert. Ein Beispiel finden Sie im folgenden Abschnitt.

1.16 Die Shell

Die Shell ist der Kommandointerpreter des UNIX-Systems. Da die Shell letztlich eine Anwendung ist, kann man sie leicht austauschen. Je nach persönlichem Geschmack kann sich jeder Anwender seine eigene Shell zuordnen lassen. Welche Shell nach dem Einloggen gestartet wird, steht in der Datei `/etc/passwd` (siehe S. 144). Man kann eine Shell aber auch jederzeit ohne Zugriff darauf direkt von der Konsole starten.

Einige Fähigkeiten der Shell sind bereits schon besprochen worden. So ist die Auswertung der Wildcards eine Funktionalität der Shell. Auch das Zeichen für die Umleitung von Ein- und Ausgabe, die Pipe und sogar die Reihenfolgebefehle für das Starten der Programme sind eigentlich Dinge, die von der Shell abhängen. Glücklicherweise halten sich die verschiedenen Shells an diesen Stellen an den von der ersten Shell, der Bourne-Shell vorgegebenen Standard.

Auch die Jobverwaltung für Prozesse wurde schon als besondere Fähigkeit der Shell genannt.

Im Kapitel über die Programmierung von Shellskripten (siehe S. 479) wird auf die Programmierung mit den Shellbefehlen eingegangen. Dort finden sich Strukturbefehle, die im Tagesgeschäft sehr hilfreich sein können. Man kann diese Befehle nämlich auch anwenden, ohne gleich ein Skript

schreiben zu müssen. So kann man beispielsweise die Schleife `for` immer dann gut einsetzen, wenn man mit mehreren Dateien bestimmte Dinge anstellen möchte. Beispiel:

```
gaston> for i in *.mett
> do
> verwurste $i >$i.neu
> rm $i
> mv $i.neu $i
> echo $i
> done
gaston>
```

Die Befehlseingabe wird so lange nicht beendet, bis das abschließende `done` nicht genannt ist, obwohl die Return-Taste gedrückt wurde. Im Beispiel nimmt die Variable `i` alle Dateinamen des aktuellen Verzeichnisses nacheinander an, deren Maske auf `*.mett` passt. Mit dem Programm `verwurste` wird jede der Dateien nacheinander erst in eine neue Datei umgewandelt. Dann wird das Original gelöscht und schließlich die neu entstandene Datei in den Originalnamen umbenannt. Ein solches Szenario würde sich anbieten, wenn die Dateien derart groß sind, dass eine parallele Bearbeitung vielleicht die Kapazität der Festplatte übersteigen würde. Nähere Informationen zur Syntax von `for` finden Sie auf Seite 490.

1.16.1 alias

Außer der ersten Bourne-Shell kennen alle Shells den Befehl `alias`. Man kann damit ein neues Kommando, ein so genanntes Alias, für komplexere Befehle geben. Eine der einfacheren Umsetzungen ist `ll` für das oft benötigte `ls -l`. Dazu wird definiert:

```
alias ll 'ls -l'
```

Nun wird jedes Mal, wenn `ll` als Befehl an der Konsole eingetippt wird, diese Zeichenkette durch `ls -l` ersetzt. Bei vielen Systemen findet man bereits einen solchen alias auf `ll` oder `l` vorinstalliert. Es ist durchaus möglich, `ll` auch Parameter anzuhängen.

Um einen Alias wieder aufzuheben, wird der Befehl `unalias` verwendet.

Weil man mit `alias` viele neue Befehle definieren kann, möchte man manchmal auch wissen, was der Befehl, den man gerade ausführt, wirklich ist.

```
gaston> type ll
ll is aliased to 'ls -l'
```

Handelt es sich bei dem Befehl nicht um einen Alias, wird der Pfad der Programmdatei angezeigt. Mit dem Befehl `file` (siehe S. 56) kann man dann weiter feststellen, ob es sich um ein Skript oder ein kompiliertes Programm handelt.

1.16.2 Startupdateien der Shell

Beim Starten einer Shell werden automatisch zu Anfang zwei Dateien ausgeführt. Dabei steht eine dieser Dateien im Verzeichnis `/etc` und wird vom root für alle Benutzer vorgegeben. Für die C-Shell heißt diese Datei **`csch.cshrc`** und für die anderen Shells heißt sie **`profile`**. Diese zentrale Datei gilt für alle Benutzer und sorgt für eine gleiche Startumgebung.

Grenzen setzen Mit Hilfe dieser Dateien werden Standardvariablen wie `PATH` vorbesetzt. Hier werden aber auch Begrenzungen eingerichtet, typischerweise mit dem Befehl `ulimit` (siehe S. 149). Diese Begrenzungen kann man sich als Anwender mit `ulimit -a` anschauen.

Aus dem Heimatverzeichnis des Anwenders wird ein Shellskript gestartet, das vor allem Einstellungen beinhaltet, die der Benutzer nach eigenem Geschmack einstellen kann. Für die C-Shell ist das die Datei **`.cshrc`** und für die anderen Shells **`.profile`**.

1.16.3 Bourne-Shell (sh) und POSIX

Die Bourne-Shell ist die erste Shell gewesen und wurde mit `sh` aufgerufen. Der POSIX-Standard fordert nach POSIX.2, dass jedes konforme Betriebssystem durch den Aufruf von `sh` eine POSIX-konforme Shell startet. Da POSIX ein Mindeststandard ist, findet man dort manchmal weit über diesen Standard hinaus gehende Shells. Auf jeden Fall hat damit die Bourne-Shell beinahe nur noch historische Bedeutung. Da aber die `sh` nach POSIX immer verfügbar ist, ist es aus Portabilitätsgründen sinnvoll, Shellskripten (siehe S. 479) für diese Shell zu schreiben, wenn sie auf unterschiedlichen UNIX-Rechnern laufen sollen. Es ist aber deswegen nicht notwendig, diese Shell auch als Anwendershell zu benutzen, da das Shellskript die benötigte Shell selbst starten kann.

Kompatibel, aber unkomfortabel Die Möglichkeiten der Bourne-Shell sind ein wenig eingeschränkt, wenn man sie mit den modernen Shells vergleicht. Insbesondere ihren Komfort beim Editieren der Kommandos könnte man mit dem Begriff »archaisch« umschreiben. Ein Zurückholen der letzten Kommandos ist beispielsweise

nicht möglich. Aus diesem Grund ist diese Shell höchstens noch aus Kompatibilitätsgründen vorhanden. Für die normale Arbeit wird man normalerweise eine modernere Shell verwenden. Will man eine möglichst hohe Kompatibilität haben, wird man die Korn-Shell oder die bash verwenden.

1.16.4 Korn-Shell (ksh)

Die Korn-Shell ist eine Weiterentwicklung der Bourne-Shell mit höherem Komfort. So kennt sie Aliase und die Jobverwaltung mit jobs, bg und fg.

Beim Start werden die Dateien `/etc/profile` und die Datei `.profile` aus dem Heimatverzeichnis gestartet.

Beginnt ein Verzeichnis mit einer Tilde `~`, so wird diese durch den Pfad des Heimatverzeichnisses ersetzt. `~/test` wird die Shell zu `/home/arnold/test` auflösen, wenn das Heimatverzeichnis `/home/arnold` ist.

Namensauflösungen

Durch zweimaliges Tippen der ESC-Taste versucht die Korn-Shell, den Datei- oder Pfadnamen des angefangenen Wortes anhand der erreichbaren Datei- bzw. Verzeichnisnamen zu vervollständigen.

Das Größerzeichen leitet die Standardausgabe (`stdout`) auf eine Datei um. Die Umleitung der Standardfehlerausgabe (`stderr`) erfolgt, indem eine 2 vor das Größerzeichen gesetzt wird. Sollen `stdout` und `stderr` in die gleiche Datei umgeleitet werden, wird `2>&1` verwendet.

Umleitung der Ausgabe

Befehl	Wirkung
cc haus.c > out	Umleitung der Standardausgabe, Fehler am Terminal
cc haus.c 2> out	Umleitung der Fehler, Standardausgabe am Terminal
cc haus.c 2>&1 out	Umleitung der Ausgabe und der Fehler nach out

Tabelle 1.28 Umleitungen

Besonderheiten bei Variablen

Der Befehl `export` dient dazu, Variablen an Nachfolgeprogramme weiterzugeben. Variablen werden für die aktuelle Shell definiert. Wird ein Programm von der Shell aus gestartet, werden die Variablen nicht weitergegeben. Das kann besonders lästig sein, wenn die Variablen eigentlich das Verhalten des gestarteten Programms ändern sollen. Damit die Variable und ihre Inhalte an alle Kindprozesse weitergegeben werden, muss die Variable exportiert werden. Als Beispiel dient ein Programm, das über die Variable `DATA` den Pfad und Dateinamen seiner Daten übermittelt bekommt.

export

```
gaston> DATA=/home/arnold/my/database
gaston> export DATA
gaston> auswertung
```

let lässt rechnen Die Korn-Shell kann mathematische Ausdrücke berechnen und in Variablen ablegen. Dazu stellt man den Befehl `let` einer Variablenzuweisung voran. Als Operanden können `+`, `-`, `*`, `/` und `%` (als Modulo) verwendet werden. Auch Klammern werden korrekt interpretiert. Es darf allerdings kein Leerzeichen in dem Ausdruck stehen. Besonders praktisch ist die Fähigkeit, mit verschiedenen Zahlensystemen zu arbeiten. Dazu wird einer Zahl die Basis, gefolgt von einem Hashzeichen (`#`), vorangestellt. Die Dualzahl `2#11` entspricht einer dezimalen 3, und um die Hexadezimalzahl `1a` darzustellen, wird `16#1a` oder `16#1A` verwendet. Beispiel:

```
gaston> let wert=45+5
gaston> echo $wert
50
gaston> wert=45+5
gaston> echo $wert
45+5
gaston> let wert=16%5
gaston> echo $wert
1
gaston> let wert=(1+3)*2
gaston> echo $wert
8
gaston> let wert=16#1a
gaston> echo $wert
26
gaston>
```

Im Beispiel ist zu sehen, dass ohne die Verwendung von `let` die Variable den Ausdruck als Text übernimmt. Die Korn-Shell kennt auch Modulo-Rechnung und Klammern. In den letzten beiden Beispielen wird die Umrechnung von hexadezimalen Werten demonstriert.

History

History mit `fc` Die Korn-Shell merkt sich standardmäßig die letzten 128 Kommandos in der Datei `.sh_history` des Heimatverzeichnisses. Die Bearbeitung der Kommandos erfolgt mit dem Befehl `fc`. Ohne weitere Parameter aufrufen, startet `fc` den Systemeditor (meist `vi`). Darin findet sich das zuletzt

eingeebene Kommando. Im Editor kann man das Kommando verändern und speichern. Dadurch wird die geänderte Version ausgeführt.

Wird als Argument von `fc` eine Zeichenkette angegeben, wird der letzte Befehl in den Editor geladen, der mit dieser Zeichenkette beginnt. Mit der Option `-e` kann ein Kommando direkt verändert werden, ohne den Editor zu belästigen. Wieder kann mit dem Argument nach der letzten Zeile gesucht werden, die mit dieser Zeichenkette beginnt. Mit einer Zuweisung kann ein Teilstring dieses Kommandos ersetzt werden. In der Praxis sieht das so aus:

```
gaston> let wert=16#1A
gaston> echo $wert
26
gaston> fc -e -
echo $wert
26
gaston> fc -e - 1A=22 let
let wert=16#22
gaston>
```

Im letzten `fc` wird das letzte Kommando gesucht, das mit »let« beginnt und innerhalb des Kommandos die Zeichenkette `1A` durch `22` ersetzt.

Mit Erleichterung werden Sie zur Kenntnis nehmen, dass Sie, anstatt mit `fc` zu arbeiten, die Editierkommandos des `vi` auf die Kommandoebene holen können. Der folgende Befehl schaltet dies ein:

```
set -o vi
```

Danach schaltet die ESC-Taste in den Editiermodus um. Man kann nun mit `+` und `-` die vergangenen Zeilen wieder heranziehen, bis man zu dem Befehl gelangt, den man ausführen möchte. Man kann sogar mit dem Schrägstrich in den bisherigen Kommandos suchen. Hat man den gewünschten Befehl gefunden, kann man ihn noch modifizieren. Innerhalb der Zeile kann man mit `i` einfügen und mit `d` löschen. Das Bewegen innerhalb der Zeile funktioniert mit der Leertaste und dem Backspace, aber auch wortweise mit `w` und `b`. Wer mit dem `vi` umgehen kann, wird sich wohlfühlen.

**Kommandos
editieren wie im vi**

```
set -o emacs
```

Die Benutzer des `emacs` werden sich schon gedacht haben, dass man auch dessen Kommandos zur Bearbeitung der History verwenden kann. Wichtig sind vor allem die folgenden Befehle:

**Kommandos
editieren wie im
emacs**

Taste	Funktion
ctrl-P	zeilenweises Rückwärtsblättern der Kommandos
ctrl-N	zeilenweises Vorwärtsblättern der Kommandos
ctrl-B	mit dem Cursor nach links
ctrl-F	mit dem Cursor nach rechts
ctrl-A	an den Anfang der Zeile
ctrl-E	an das Ende der Zeile
ctrl-D	löscht das Zeichen unter dem Cursor
ctrl-K	lösche bis an das Ende der Zeile
ctrl-R	inkrementelles Suchen

Tabelle 1.29 emacs-Kommandotasten

1.16.5 C-Shell (csh)

Die C-Shell ist vor allem für Programmierer entwickelt worden. Ihre Skriptsprache unterscheidet sich erheblich von anderen Shells und ähnelt der Sprache C. Die Leistungsfähigkeit der Skriptsprache ist heutzutage nicht mehr so relevant, da man Perl oder Tcl/Tk heranzieht, wenn man eine leistungsstärkere Skriptsprache als die der Shell benötigt. Unabhängig davon hat die C-Shell immer noch ihre Freunde. Insbesondere die modernere Variante tcsh, die einige Fähigkeiten der bash (siehe S. 107) besitzt, ist durchaus verbreitet. So findet sie sich auch auf Mac OS X als Standard-Shell.

Die Startdatei der C-Shell heißt **.cshrc**. Statt der Datei **.profile** wird bei einer Login Shell die Datei **.login** gestartet. Neu gegenüber der Bourne-Shell ist, dass es auch eine Datei gibt, die beim Ausloggen gestartet wird. Sie heißt konsequenterweise **.logout**.

Durch ESC versucht die C-Shell den angefangenen Dateinamen anhand der im Verzeichnis existierenden Dateinamen automatisch zu vervollständigen.

Umgebungsvariablen

Die C-Shell unterscheidet besonders genau zwischen Shellvariablen und Umgebungsvariablen. Letztere werden mit dem Befehl **setenv** gesetzt und mit **printenv** angezeigt.


```
setenv VARIABLE wert
```

Dagegen werden die lokalen Shellvariablen mit `set` gesetzt und durch Voranstellen eines Dollarzeichens ausgelesen. Dabei wird auf die Umgebungsvariable nur dann zurückgegriffen, wenn es keine Shellvariable gibt. So können Umgebungsvariablen und Shellvariablen sogar unabhängig nebeneinander existieren, wie man im folgenden Beispiel sieht:

```
gaston> setenv huhu 3
gaston> echo $huhu
3
gaston> set huhu=5
gaston> echo $huhu
5
gaston> printenv huhu
3
gaston> ps
```

Beim Umleiten des Fehlerausgabekanals (**stderr**) in eine Datei muss bei der C-Shell auch der normale Ausgabekanal (**stdout**) in diese Datei umgeleitet werden. Das Zeichen für die gemeinsame Umleitung ist eine Kombination aus Größerzeichen und kaufmännischem Und (>&).

Umleitung

```
cc -o moin moin.c >& outerr
```

Dieser Mechanismus gilt ebenso bei einer Pipe. Wird dem senkrechten Strich ein & angehängt, geht auch **stderr** durch die Pipe. Anders ausgedrückt geht die **stderr** bei der C-Shell entweder auf den Bildschirm oder folgt der **stdout** durch Angabe des kaufmännischen Und (&) hinter dem Umleitungszeichen.

History

Die C-Shell verwendet das Ausrufezeichen als Historyzeichen. Daneben werden auf den neueren Systemen normalerweise die Cursortasten unterstützt.

Zeichen	bewirkt...
!!	ruft die letzte Zeile noch einmal auf
!5	ruft die fünftletzte Zeile noch einmal auf
!abc	ruft die letzte Zeile auf, die mit abc beginnt
!?abc	ruft die letzte Zeile auf, die abc enthält
!\$	verwendet das Argument der letzten Zeile an dieser Stelle

Tabelle 1.30 C-Shell-History

Teile des letzten Kommandos können ersetzt werden. Dazu folgendes Beispiel:

```
hpsrv 2: lx hel*
lx: Command not found.
hpsrv 3: ^lx^ls
ls hel*
hello.cpp
hpsrv 4:
```

Das fehlerhafte lx wird durch das korrekte ls ausgetauscht. Nach Return wird der korrigierte Befehl gestartet. Im folgenden Beispiel wird noch einmal der Befehl geholt, der mit ls beginnt. Im Anschluss wird das Argument des letzten Kommandos im neuen Kommando wieder verwendet.

```
hpsrv 4: !ls
ls hel*
hello.cpp
hpsrv 5: echo !$
echo hel*
hello.cpp
hpsrv 6:
```

Promptgestaltung Statt der Umgebungsvariablen PS1 für den Prompt verwendet die C Shell die Variable prompt, die mit set gesetzt wird:

```
set prompt="%\!:"
```

Dieser Befehl führt dazu, dass vor jedem Kommando eine Nummer steht, auf die man sich in der History beziehen kann.

1.16.6 Bourne-Again-Shell (bash)

bash ist die Abkürzung für *bourne again shell*. In ihrer Kommandostruktur setzt sie also auf der alten Bourne-Shell auf, fügt ihr aber diverse Fähigkeiten hinzu, die man auch in anderen Shells wie beispielsweise der Korn-Shell (ksh) oder der C-Shell (csh) findet.

Die bash ist die Standard-Shell der Linux-Systeme, findet sich aber auch auf anderen UNIX-Derivaten, selbst für OS/2 oder MS Windows kann man die bash bekommen. Unter Linux gibt es keine sh mehr. Der Standardeintrag `/bin/sh` ist ein Link auf die bash. Da die bash eine Obermenge der sh ist, ist eine separate Bourne-Shell auch nicht erforderlich.

**bash ist Standard
unter Linux**

Ablauf der Startdateien

Nachdem die Datei `/etc/profile` abgearbeitet wurde, wird die Datei `.profile` im Heimatverzeichnis gestartet. Innerhalb der Datei `.profile` wird die Ausführung der Datei `.bashrc` gestartet, sofern sie vorhanden ist. Am Ende der Datei `.bashrc` wird die Datei `.alias` gestartet, in der typischerweise die Alias-Definitionen gesammelt werden.

History

Normalerweise ist die bash so installiert, dass mit den Pfeiltasten durch die bisherigen Kommandos navigiert werden kann. Wenn es um mächtigere Kommandos geht, werden die Tastenkombinationen des emacs verwendet. Wie bei der Korn-Shell kann man mit `set -o vi` die Tastenkombinationen des vi aktivieren. Zu guter Letzt finden auch Freunde der C-Shell ihre History-Funktionen mit dem Ausrufezeichen implementiert.

Funktionstasten

Mit den Pfeiltasten kann man vor und rückwärts durch die Befehle blättern. Mit den Links- und Rechtspfeiltasten kann man den Cursor innerhalb des Befehls positionieren. `Ctrl-A` bewegt den Cursor an den Anfang der Zeile, `ctrl-E` bringt ihn zum Ende. Backspace funktioniert wie erwartet. Das Löschen des Zeichens unter dem Cursor erfolgt mit `ctrl-D` oder mit der Taste Delete. Die inkrementelle Suche funktioniert wie beim emacs (siehe S. 70). Bei jedem Tastendruck wird in der History nach einem passenden Befehl gesucht. Dabei schlägt bash immer den nächsten passenden Befehl vor.

```
gaston>  
(reverse-i-search)'ps': ps -axwww
```

Taste	Funktion
ctrl-A	an den Anfang der Zeile
ctrl-E	an das Ende der Zeile
ctrl-D	löscht das Zeichen unter dem Cursor
ctrl-K	löscht bis an das Ende der Zeile
ctrl-R	inkrementelles Suchen

Tabelle 1.31 Funktionstasten in bash

Dateinamenergänzung

Wenn ein Dateiname mit den ersten Buchstaben eingegeben ist, kann man ihn mit der TAB-Taste vervollständigen, wenn er eindeutig ist. Ist der Name nicht eindeutig oder existiert kein Dateiname mit diesen Zeichen, piept es. Durch nochmaliges Tippen von TAB wird die Liste aller passenden Namen angezeigt.

Die Ergänzung funktioniert nicht nur bei den Parametern. Auch beim Aufruf eines Befehls können die ersten Zeichen eingegeben werden und mit der TAB-Taste vervollständigt werden. Dabei werden alle ausführbaren Dateien angeboten, die über den Pfad der Variable PATH erreichbar sind.

Best of...

Wie schon erwähnt, wurde in der bash alles realisiert, was in anderen Shells bemerkenswert ist. Das beginnt bei der Korn-Shell mit dem Befehl `fc` bis zur Auswertung arithmetischer Ausdrücke mittels `let`, und die Liebhaber der C-Shell finden ihre Ausrufezeichen wieder. Das ist wohl der Grund, warum die bash inzwischen sehr weit verbreitet ist.

1.17 Ausgaben auf dem Drucker

In diesem Abschnitt wird das Drucken aus Anwendersicht betrachtet. Unter dem Aspekt der Installation und Konfiguration wird das Thema Drucken noch einmal im Administrationskapitel behandelt (siehe S. 209).

Es gibt zwei Drucksysteme unter UNIX. Dies ist historisch durch die Aufteilung zwischen BSD-UNIX aus dem universitären Bereich und dem AT&T-UNIX entstanden. Welches System auf dem eigenen Rechner installiert ist, erkennt man am Aufruf. Funktionieren die Aufrufe `lpr`, `lpq` und `lprm`, ist es offensichtlich ein BSD-System.

1.17.1 BSD-Unix: lpr, lpq und lprm

Das BSD-Drucksystem ist inzwischen recht weit verbreitet, da es Netzwerkdrucker unterstützt. Dieser Standard ist bereits in andere Betriebssysteme als UNIX übernommen worden, sodass er sich inzwischen zu einem Quasistandard für TCP/IP-Drucker entwickelt hat. Dass Mac OS X das BSD-System verwendet, verwundert nicht, immerhin basiert es auf FreeBSD. Aber auch Linux, das sich in vielen Dingen an System V hält, verwendet primär das BSD-System.

Start des Druckauftrags

Der Server des Druckdienstes heißt `lpd`. Um als Anwender Dateien in das Spoolverzeichnis des `lpd` zu schreiben, wird das Frontendprogramm `lpr` verwendet. Will man eine Datei auf den Standarddrucker ausgeben, reicht der Befehl:

```
lpr meinedatei
```

`lpr` wird die Datei an den `lpd` senden und ihm mitteilen, welches der Standarddrucker ist. Die Information, welcher Drucker der Standarddrucker ist, entnimmt `lpr` der Umgebungsvariablen `PRINTER`. Daraufhin wird `lpd` den Druckauftrag zum Spoolen einstellen und nach und nach auf dem Drucker ausgeben.

PRINTER

Typischer ist es jedoch, dass die Druckdatei als solche gar nicht existiert, sondern der Text vor dem Druck aufbereitet oder erst erzeugt wird. Dann wird der Druck an den `lpr` per Pipe übermittelt. Beispielsweise wird eine Textdatei meist vor dem Druck etwas geschönt und nach PostScript konvertiert. Standardmäßig geht UNIX davon aus, dass Drucker PostScript beherrschen.

**Drucken aus der
Pipe**

```
pr -h "/etc/printcap vom 3.3.2001" /etc/printcap | lpr
```

Im Beispiel wird die Druckerkonfigurationsdatei mit einer Titelzeile ausgedruckt. Das Kommando `pr` (siehe S. 111) formatiert ASCII-Dateien vor. Das Ergebnis geht auf **stdout** und von dort über eine Pipe an `lpr`, der den Druckauftrag erteilt. Man könnte auch auf diese Weise eine aktuelle Prozessliste ausgeben:

```
ps -alx | pr -h "ps um 14:13" | lpr
```

Die Prozessliste wird mit `ps` erzeugt, mit `pr` formatiert und wandert dann zum Drucker.

Umgang mit mehreren Druckern

Häufig steht aber nicht nur ein einziger, sondern mehrere Drucker zur Verfügung, und so wird in einem Parameter für den `lpr` angegeben, welcher der Drucker verwendet werden soll. Angenommen, es gäbe noch einen Farblaserdrucker, der sinnigerweise `farblaser` getauft wurde. Der Aufruf lautet unter Linux oder Solaris²⁸:

```
a2ps meinedatei | lpr -Pfarblaser
```

Der Name des Druckers wird in der Datei `/etc/printcap` (siehe S. 211) definiert. Diese enthält auch die Information, ob der Drucker direkt an dieser Maschine angeschlossen ist oder ob er über das Netz erreichbar ist.

Druckkontrolle

Druckstatus: `lpq`

Mit dem Befehl `lpq` kann man sich die laufenden Druckjobs anzeigen lassen. Verwendet man einen anderen als den Standarddrucker, wird der gleiche Parameter `-P` wie beim `lpr` verwendet, um die Druckwarteschlange zu betrachten. Alle eingestellten Druckaufträge bekommen eine Nummer. Diese Nummer ist wichtig, wenn man einen Druckauftrag wieder entfernen will.

Stornieren: `lprm`

Mit `lprm` und besagter Nummer kann man einen eingestellten Druckauftrag wieder aus der Warteschlange entfernen. Das funktioniert eventuell nicht mehr, wenn der Druckdämon den Auftrag bereits in der Bearbeitung hat.

1.17.2 AT&T: `lp`, `lpstat` und `cancel`

`lp -d` Drucker

Beispielsweise verwenden AIX und SCO von Haus aus das AT&T-Drucksystem. Mit dem Kommando `lp` werden auf AT&T-Systemen Druckaufträge abgesetzt. Um einen anderen als den Standarddrucker anzusprechen, wird die Option `-d`, gefolgt vom Druckernamen, verwendet.

Druckkontrolle: `lpstat` und `cancel`

Ausgabe von `lpstat`

Der Befehl `lpstat` gibt eine Liste über den Status der verschiedenen Drucker aus. Beispiel:

Queue	Dev	Status	Job	Files	User	PP	%	Blks	Cp	Rnk
-----	-----	-----	---	-----	-----	--	--	----	---	---
lp0	lp0	RUNNING	918	STDIN.14846	willemer	5	43	10	1	1

²⁸ Unter SCO heißt die Option `-d` statt `-P`. Bei Fremdsystemen wie MS Windows oder OS/2 muss noch zusätzlich der Servername mit `-S` angegeben werden.

```

        QUEUED  919 STDIN.14593  willemer          3   1   2
        QUEUED  920 maskexpj    wagener           2   1   3
        QUEUED  921 STDIN.14596  willemer          2   1   4
        QUEUED  922 STDIN.21790  willemer          1   1   5
        QUEUED  923 lunget       wagener           3   1   6
bsh    bshde READY

```

Um einen Text aus der Warteschlange zu entfernen, wird der Befehl `cancel` verwendet. Es ist natürlich nur möglich, eigene Druckaufträge wieder zu entfernen. Fremde Aufträge kann nur der jeweilige Besitzer oder der Systemverwalter beseitigen.

**cancel storniert
Druckaufträge**

1.17.3 Druck formatieren: `pr` und `a2ps`

Das Programm `pr` formatiert Textdateien seitenweise. Dies ist vor allem als Vorbereitung für den Ausdruck der Dateien hilfreich. Bei der Ausgabe werden das Datum, der Name der Datei und eine Seitennummerierung als Kopfzeile angelegt. Der Befehl hat die Optionen:

Option	Wirkung
<code>-t</code>	keine Kopf- und Fußzeilen
<code>-Zahl</code>	erzeugt einen Ausdruck mit <i>Zahl</i> Spalten
<code>-h text</code>	Als Kopfzeile wird <i>text</i> verwandt

Tabelle 1.32 Optionen von `pr`

Ein typischer Aufruf lautet:

```
pr -h "(C) Arnold Willemer" meintext | lpr
```

Um reine Textdateien auszugeben, gibt es auf vielen Systemen das Programm `a2ps`. Der Name bedeutet etwa »ASCII to PostScript«. Dabei wird der Text zweispaltig mit Überschrift und Seitennummer formatiert und dann auch gleich über den Standarddrucker ausgegeben. Das Programm `a2ps` erhalten Sie unter:

a2ps

<http://www.inf.enst.fr/~demaille/a2ps>

1.18 Zeitversetztes Arbeiten

UNIX bietet die Möglichkeit, Befehle zeitversetzt auszuführen. Für regelmäßig wiederkehrende Arbeiten gibt es die `crontab`. Soll ein Job genau einmal ausgeführt werden, verwendet man das Kommando `at`.

Ausgaben in die Mail Da zum Ausführungszeitpunkt die aufrufende Shell mit großer Wahrscheinlichkeit nicht vorhanden ist, werden die Ausgaben der zeitversetzt gestarteten Prozesse dem Aufrufer per Mail zugestellt.

1.18.1 Die aktuelle Zeit

Bevor man aber Aufträge in die Nacht verlegt, sollte man kontrollieren, welche Uhrzeit der Computer derzeit zu haben glaubt. Dabei hilft der Befehl `date`, der auch die Uhrzeit liefert.

```
gaston > date
Don Jan  3 20:34:44 CET 2002
```

Vom Administrator kann der Befehl auch verwendet werden, um die Uhrzeit zu setzen. Dabei wird direkt hinter dem Befehl das neue Datum in direkter Ziffernfolge ohne Sonderzeichen gesetzt. Beispiel:

```
gaston # date 01032044
Don Jan  3 20:44:00 CET 2002
```

Die acht Ziffern werden nach dem Muster `MMDDhhmm` angegeben, also je zwei Stellen für Monat, Tag, Stunde und Minute. Das Jahr kann mit zwei oder vier Stellen angehängt werden.

Das Zeitformat ist änderbar Nützlich ist die Möglichkeit, die Ausgabe von `date` zu formatieren. Dazu wird hinter einem Pluszeichen eine Zeichenkette (meist in Anführungszeichen) mit einem Formatmuster angegeben. Das Datum in deutscher Schreibweise würde man mit dem folgenden Befehl erzeugen:

```
gaston > date +"%d.%m.%Y"
03.01.2002
```

Kürzel	Bedeutung
%H	Stunden (00-23)
%M	Minuten (00-59)
%S	Sekunden (00-59)
%d	Tag im Monat (00-31)
%m	Monat (00-12)
%y	zweistelliges Jahr (00-99)
%Y	die vollständige Jahreszahl

Tabelle 1.33 Formatzeichen von `date`

Der besondere Nutzen der Formatierung wird deutlich, wenn man Protokolldateien Namen geben will, die mit ihrer Entstehung zu tun haben, wie man es in Skripten oder crontabs braucht. Der Dateiname im folgenden Beispiel würde am 3. Januar **proto03** heißen.

```
proto`date +%d`
```

Auf einem PC unter Linux wird die Uhrzeit durch eine Hardware-Uhr gehalten, die batteriegespeist auch beim Ausschalten des Geräts weiterläuft. Durch den Aufruf von `date` wird diese Uhr allerdings nicht gesetzt. Um die Systemuhrzeit auf diesen Chip zu übernehmen, ruft der Administrator den folgenden Befehl auf:

Hardwareuhr

```
hwclock --systohc
```

1.18.2 Regelmäßige Arbeiten: crontab

Jeder Anwender kann sich eine **crontab** erstellen, die wiederkehrende Arbeiten automatisch zu festgelegten Zeiten definiert. Das Systemprogramm `cron` arbeitet diese Tabelle ab und startet die darin aufgeführten Programme. Die **crontab** wird in einer Datei im Verzeichnis `/var/spool/cron` abgelegt. Da dieses Verzeichnis aber nur von Administratoren genutzt werden darf, wird die **crontab** mit dem Kommando `crontab` gepflegt. Im ersten Schritt sollte man prüfen, ob man vielleicht bereits eine eingerichtete **crontab** besitzt. Dies kann man recht einfach feststellen:

```
crontab -l
```

Gibt man `crontab` ohne Parameter ein, wartet das Programm, bis man die `crontab` von der Tastatur direkt eingetippt hat. Man beendet die Eingabe mit `ctrl-D`. Es ist allerdings wesentlich entspannender, wenn auch nicht so imposant, wenn man einen Editor verwendet und die **crontab** zunächst in eine Datei sichert. Anschließend kann man durch Umleitung der Datei in die Eingabe von `crontab` bequem die Tabelle definieren. Um eine **crontab** zu aktualisieren, ist der übliche Vorgang, dass man die bisherige **crontab** in eine Datei umleitet, sie mit einem Editor anpasst und dann als Eingabedatei in die `crontab` schiebt. Da die Datei anschließend nicht mehr gebraucht wird, kann man sie löschen.

crontab verändern

```
crontab -l > meincrntab
vi meincrntab
crontab < meincrntab
rm meincrntab
```

Die **crontab** ist eine Textdatei, in der zunächst in mehreren Spalten das Wiederholungsintervall festgelegt wird. In der letzten Spalte wird das zu startende Kommando eingetragen. Das Beispiel hat der besseren Lesbarkeit halber zu Anfang zwei Kommentarzeilen. Die müssen in der crontab nicht stehen.

Listing 1.1 Beispiel einer crontab

```
# Minute Stunde Tag(Monat) Monat Tag(Woche) Kommando
# (0-59) (0-23) (1-31) (1-12) (1-7; 1=Mo)
0      4      *      *      *      program1
0      9      1,15    *      *      program2
0      2      *      *      1,2,3,4,5 program3
```

Die ersten fünf Spalten beschreiben den Startzeitpunkt und die sechste das auszuführende Kommando. Ein Stern bedeutet, dass die Ausführung stattfindet, egal welchen Wert die Kategorie hat. Eine von Kommata getrennte Zahlenkolonne gibt mehrere Zeiten an. Aus dem Beispiel ergibt sich:

- ▶ program1 wird um 4:00 Uhr an jedem Tag gestartet.
- ▶ program2 wird um 9:00 Uhr am 1. und 15. jeden Monats gestartet.
- ▶ program3 wird um 2:00 Uhr an jedem Montag, Dienstag, Mittwoch, Donnerstag und Freitag gestartet.

Beim Kommando crontab kann mit der Option -u der Benutzer angegeben werden, dessen **crontab** bearbeitet werden soll. Dabei ist das Verändern fremder **crontabs** natürlich das Privileg des Superusers.

1.18.3 Zeitversetzter Job: at

Der Befehl at führt ein Kommando zu einem angegebenen Zeitpunkt einmal aus. Als Parameter wird die Uhrzeit angegeben. Anschließend werden die Kommandos zeilenweise eingegeben und durch ctrl-D beendet. Zum Beispiel:

```
# at 16:00
at>ping comeback
at> ^D
#
```

Zeitjobs verwalten Um 16:00 wird ein ping an den Rechner comeback abgesetzt. Um eine Liste aller abgesetzten at-Befehle anzusehen, gibt es den Befehl atq. In

dieser Liste stehen auch Nummern, über die die Befehle wieder mit `atrm` zu löschen sind.

Sollte im Beispiel 16.00 Uhr bereits überschritten sein, wird der `ping` erst morgen um diese Zeit ausgeführt. Man kann auch das Datum der Ausführung festlegen. Dieses hat das Format `DD.MM` oder `DD.MM.YY`, dabei kann das Jahr auch vierstellig sein. Sehr praktisch ist es auch, dass man eine Zeitdifferenz angeben kann. Das Beispiel wird eine Minute nach dem Start des Kommandos beginnen.

```
at now + 1 minute
```

1.19 Diskettenlaufwerke

Disketten werden auf drei Arten in UNIX-Systemen verwendet:

1. mit einem UNIX-Dateisystem. Sie wird dann mit dem Befehl `mount` eingebunden.
2. als `tar`-Archiv
3. zum Austausch mit anderen Betriebssystemen. Dazu gibt es spezielle Programme, die mit den MS-DOS-Dateisystemen auf Disketten umgehen können.

1.19.1 Formatieren und Beschreiben

Das Formatieren einer Diskette ist bei UNIX-Systemen unabhängig vom Anlegen eines Dateisystems. Der Grund liegt darin, dass eine Diskette unter UNIX auch ohne Dateisystem verwendbar ist. Man kann sie beispielsweise als Device für ein `tar` verwenden. Die Aufrufe zum Formatieren einer Diskette unterscheiden sich etwas von System zu System. Unter Linux beispielsweise sind es drei Schritte vom Entnehmen der neuen Diskette aus der Verpackung bis zu dem Zeitpunkt, da sie beschrieben werden kann:

```
fdformat /dev/fd0h1440
mkfs -t ext2 -c /dev/fd0h1440
mount /dev/fd0h1440 /floppy
```

Auf einer Sun unter Solaris verwendet man:

```
fdformat
newfs /dev/rdiskette
```

Danach kann die Diskette wie eine Festplatte verwendet werden.

Unter SCO lautet der Befehl `format`. Das Device muss nicht angegeben werden, da es bereits nach der Standardinstallation korrekt in der Datei `/etc/default/format` steht.

1.19.2 mount und eject

Auch Disketten können und müssen per `mount` in den Verzeichnisbaum eingebunden werden. Nach der Arbeit mit der Diskette ist die Versuchung groß, sie einfach wieder aus dem Computer zu holen. Aber sie muss vorher per `umount` wieder freigegeben werden. Damit niemand der Versuchung erliegt, ist bei den Diskettenlaufwerken von Sun beispielsweise gar kein Auswurfknopf am Laufwerk vorhanden. Man muss die Diskette mit dem Befehl `eject /dev/diskette` auswerfen lassen.

Mac OS X verwendet Automount, wenn auf wechselbare Speichermedien zugegriffen wird. Bei diesem Verfahren werden den Geräten Standardpfade zugewiesen, auf denen sie eingehängt werden. Wird ein solcher Pfad bei einem Zugriff verwendet, wird automatisch das dazu gehörige Gerät per `mount` eingebunden. Sobald der Pfad eine bestimmte Zeit nicht mehr im Zugriff ist, wird das Gerät wieder ausgekoppelt. Automount ist auch auf anderen UNIX-Systemen verfügbar und kann dort analog installiert werden (siehe dazu S. 334).

1.19.3 tar und sync

Da man auf Disketten meist nur Dateien zum Transport oder zur Sicherung abspeichern will, braucht man nicht unbedingt ein Dateisystem anzulegen. Meist ist es sinnvoller, die Disketten per `tar` zu beschreiben. Das ist insbesondere deswegen von Vorteil, weil `tar` auch in der Lage ist, die Daten auf mehrere Medien zu verteilen. Es ist so auch problemlos möglich, Dateien zu übernehmen, die größer als die Kapazität einer Diskette sind.

Disketten, die mit `tar` bearbeitet werden, werden nicht per `mount` in den Verzeichnisbaum eingebunden. Während auf einer Sun die Kontrolle über den `eject` gewährleistet ist, ist beim PC der Anwender selbst dafür verantwortlich, dass die Daten vollständig auf der Diskette angekommen sind. Ein Abwarten, bis die Kontrollleuchte ausgeht, ist obligatorisch, und man sollte immer den Befehl `sync` verwenden, der dafür sorgt, dass die Puffer sofort auf die Medien geschrieben werden.

1.19.4 MS-DOS-Disketten

Trotz all ihrer Nachteile stellen MS-DOS-Disketten einen gewissen systemübergreifenden De-facto-Standard dar. Nahezu jedes Betriebssystem

kann Dateien von einer solchen Diskette lesen oder sogar schreiben. Ein Problem sind die Beschränkungen. Der Dateiname kann die Groß- und Kleinschreibung nicht unterscheiden, und der Name muss im 8.3-Format vorliegen, also aus maximal acht Zeichen, dann einem Punkt und danach noch einmal aus maximal drei Zeichen bestehen.

Linux ist mit einem Satz von Befehlen ausgestattet, der speziell für das Bearbeiten von MS-DOS-Disketten geschrieben wurde.

MS-DOS	Linux	Bedeutung
DIR A:	mdir	zeigt den Inhalt der Diskette an
COPY <i>Quelle Ziel</i>	mcopy <i>Quelle Ziel</i>	Kopieren von Dateien
DEL <i>Dateien</i>	mdel <i>Dateien</i>	Löschen von Dateien

Tabelle 1.34 Befehle für MS-DOS-Disketten

Texte, die auf den Systemen MS-DOS oder MS Windows entstanden sind, verwenden als Zeichentrenner die Kombination Carriage Return (dezimaler Zeichencode: 13) und Line Feed (dezimaler Zeichencode 10). Bei UNIX-Texten wird nur das Line Feed verwendet. Viele Programme stört das Carriage Return nicht. Aber manchmal muss es entfernt werden. Am einfachsten gelingt dies mit dem Befehl `tr`:

Zeilentrenner anpassen

```
tr -d \r <dosdatei >unixdatei
```

Je nach System gibt es auch Programme wie `dos2unix`, die solche Umwandlungen vornehmen. Müssen auch Umwandlungen der Umlaute oder sonstiger Sonderzeichen vorgenommen werden, empfiehlt sich das Programm `recode`. Um einen Text, der unter MS-DOS erstellt wurde, nach UNIX zu konvertieren, gibt man den folgenden Befehl ein:

Zeichensätze konvertieren

```
recode ibmp..lat1 <dos.txt >neu.txt
```

Dabei ist `ibmp` die Bezeichnung des Quellzeichensatzes und `lat1` die des Zielzeichensatzes. Um sich alle Zeichensätze anzeigen zu lassen, ruft man `recode` mit der Option `-l` auf. Die Liste enthält auch Zeichensätze vom Macintosh, vom Atari ST und EBCDIC vom IBM-Großrechner. Sogar in Textbeschreibungssprachen wie HTML und \TeX kann man seine Texte umcodieren. In älteren Versionen von `recode` wurde statt den beiden Punkten ein Doppelpunkt zwischen den Zeichensätzen verwendet.²⁹

²⁹ vgl. Thissen, Thomas: Umlaute auf Umwegen. iX 9/1993, S. 194–197.

1.20 CD-ROMs

ISO 9660 mit Rockridge

Das Standardformat für CDs ist ISO-9660, auch High Sierra genannt. Dieses Format kann aber mit all dem, was eine UNIX-Datei ausmacht, wenig anfangen. Bereits der Dateiname ist auf 8+3 Buchstaben in Großbuchstaben beschränkt. Das Speichern von User-ID und anderen Eigenschaften ist nicht vorgesehen. Damit man auch unter UNIX mit CD-ROMs vernünftig umgehen kann, gibt es die Rockridge-Erweiterungen. Damit lassen sich auch CDs in das Dateisystem integrieren.

Das Einbinden funktioniert wie bei der Diskette mit dem Befehl `mount`. Auch die CD muss zunächst per `umount` ausgekoppelt werden, bevor sie entnommen werden kann. Da die CD-Laufwerke inzwischen alle eine Verriegelung besitzen, sind auch die PC-Versionen von UNIX nicht mehr darauf angewiesen, dass der Anwender aufpasst, sondern können eine eingebundene CD gegen das Herausnehmen sperren.

Device des CD-Laufwerks

Ein Problem ist es, zu bestimmen, welcher Eintrag im Verzeichnis von `/dev` mit dem CD-Laufwerk korrespondiert. Bei den großen Herstellern von UNIX-Maschinen kann man den Ort der Laufwerke meist durch die mitgelieferte Dokumentation leicht erfahren. Da meist SCSI-Einheiten eingebaut sind, kann man mit Hilfe der SCSI-ID auch recht schnell auf das Device schließen. Bei ATAPI auf dem PC hängt es davon ab, an welchem Controller das Laufwerk hängt. USB wird als SCSI-Device emuliert und wird vom System als weiterer SCSI-Controller angesprochen. Wer nun glücklich ermittelt hat, welches Device zum CD-Laufwerk gehört, tut jedenfalls gut daran, einen symbolischen Link namens `cdrom` im Verzeichnis `/dev` zu erzeugen. Mit etwas Glück hat die Standardinstallation bereits einen solchen Link angelegt.

1.21 CD-Brenner

Der Kontakt mit dem CD-Brenner erfolgt in erster Linie über das Programm `cdrecord`. Es benötigt als Parameter die Information, wie der CD-Brenner mit dem Gerät verbunden ist. Am einfachsten funktioniert das mit SCSI-Laufwerken, bei denen der Controller, die SCSI-ID und die LUN angegeben wird. LUN bedeutet *logical unit*, also logische Einheit, und wird von einigen SCSI-Geräten verwendet, um Teileinheiten anzusprechen. Beispielsweise verwenden einige CD-Wechsler die LUN für den Zugriff auf die einzelnen CDs. Die LUN ist normalerweise 0, die SCSI-ID wird am Gerät eingestellt und meistens besitzt man auch nur einen Controller. In den folgenden Beispielen verwende ich meine Konfiguration, und die SCSI-ID meines Brenners ist 3.

Das Programm `cdrecord` kommt aus dem Umfeld von Linux, findet sich aber inzwischen, da es Open Source ist, auf allen großen UNIX-Plattformen. Bei Mac OS X ist es nicht verfügbar, da das Brennen von CDs direkt in den Desktop eingebunden ist.

1.21.1 Datensicherung

Da CD-Rohlinge inzwischen billiger sind als Disketten, ist ein CD-Brenner als Datensicherungsoption auf dem PC immer interessanter. Brauchbare Streamer sind teuer und auch die Medien sind nicht billig. Wenn man dann davon ausgeht, dass Magnetbänder nach einer gewissen Laufzeit nicht mehr verwendbar sind, wird die Verwendung von CDs immer nahe liegender. Das Problem der CDs liegt in ihrer geringen Kapazität von 650 MByte. Dadurch fallen sie in der Regel für die Komplettsicherung aus.³⁰

Um eine Datensicherung von der Platte auf CD zu brennen, sind zwei Schritte erforderlich. Zunächst wird mit Hilfe des Befehls `mkisofs` ein brennbarer Speicherabzug erstellt, der anschließend mit `cdrecord` auf die CD gebrannt wird.

Erstellen eines Images

Bevor ein Verzeichnis auf CD gebrannt werden kann, wird in einer Datei ein ISO-9660-Dateisystem erzeugt. Das Programm `mkisofs` erzeugt ein solches Dateisystem und legt es in einer Datei als so genanntes *Image* ab. Der Dateiname des Images wird als Parameter mit der Option `-o` angegeben. Zuletzt wird das Verzeichnis, das mit allen Unterverzeichnissen auf die CD soll, angegeben. Im folgenden Beispiel wird mit `cd` in das zu sichernde Verzeichnis gewechselt. Es folgt der Befehl zum Erzeugen des Images:

```
mkisofs -J -R -o ../image.iso .
```

Die Optionen von `mkisofs` bedeuten:

- ▶ **-o *ausgabe.iso*** Die Ausgabe erfolgt in die angegebene Datei. Standardmäßig geht die Ausgabe nach `stdout` und kann so über eine Pipe an `cdrecord` weitergeleitet werden.
- ▶ **-R** Als Ergänzung zum Format nach ISO-9660 wird das Rockridge-Format verwendet, um lange Dateinamen und Dateiattribute auf der CD zu speichern. Dies ist Standardformat von CDs im UNIX-Bereich.

³⁰ Natürlich kann man mit Hilfe eines Packers wie `gzip` auch größere Datenbestände sichern. Allerdings wird man auch damit keine Größenordnungen erreichen, wie sie bei heutigen Platten gebraucht werden.

- -J Als Ergänzung zum Format nach ISO-9660 wird das Joliet-Format verwendet, um lange Dateinamen auf der CD zu speichern. Dies ist Standardformat von CDs unter MS Windows 95 und seinen Folgeversionen. Damit ist die CD auch unter MS Windows mit langen Namen verwendbar.

Die Zielfeile befindet sich im Verzeichnis unter dem Quellverzeichnis, damit das Image nicht in die Datensicherung hineingerät.

Brennen

Das erzeugte Image wird vom Programm `cdrecord` gebrannt. Das Programm benötigt bei einer Standardinstallation root-Rechte. Sollen auch Anwender das Gerät ansprechen, was bei einer Workstation sinnvoll ist, dann kann der Administrator den Eigentümer auf root und das User-ID-Bit setzen:

```
cd /usr/bin
chown root cdrecord
chmod 4711 cdrecord
```

Für SCSI-Brenner wird die SCSI-Nummer als Parameter benötigt. Kennt man diese nicht, kann man sie mit Hilfe von `cdrecord` und der Befehlsoption `-scanbus` ermitteln. Für das Brennen wird dem Parameter `dev=` ein Tripel aus SCSI-Bus, SCSI-ID und LUN (bei Brennern typischerweise 0) zur Bestimmung des Gerätes mitgegeben:

```
cdrecord -dev 0,3,0 -speed=2 image.iso
```

Mit der Option `speed=2` wird die Brenngeschwindigkeit auf 2 gesetzt. Schneller ist mein Brenner eben nicht.³¹ Vergisst man die Option, wird die Geschwindigkeit auf 1 gesetzt, und das viele Geld für den schnellen Brenner ist umsonst ausgegeben.

Direktes Brennen durch die Pipe

Wird zum Zweck der Datensicherung ein Bereich der Festplatte gesichert, ist die Datenquelle so schnell, dass man das Image direkt an das Brennprogramm weiterleiten kann. Der Vorteil liegt darin, dass nur 4 MByte Puffer Plattenspeicher benötigt werden, statt bis zu 650 MByte für das Image.

³¹ Sollten Ihnen soeben die Tränen des Mitleids in die Augen schießen, können Sie helfen! Empfehlen Sie dieses Buch weiter, und vielleicht kann sich der Autor dann schon bald auf dem Flohmarkt einen Brenner mit 4-facher Geschwindigkeit leisten.


```
mkisofs -R /home/mydata | cdrecord speed=2 dev=0,3,0 -
```

Kopieren einer Daten-CD

Um von einer Daten-CD eine 1:1-Kopie anzulegen, greift man direkt auf das Image des Devices als Datenquelle für cdrecord zu und braucht entsprechend mkisofs nicht. Allerdings sollte das CD-Laufwerk auch deutlich schneller und kontinuierlicher lesen, als der Brenner schreibt.

```
cdrecord -v dev=0,3,0 speed=2 -isozsize /dev/cdrom
```

Gerade schnelle CD-Laufwerke brauchen manchmal länger, um auf Ihre Zielgeschwindigkeit zu kommen. In diesen Momenten kann der Datenstrom länger ausbleiben, als der Brenner verkraften kann. Das führt zu einem Buffer-Underrun. Die Anforderung an das Laufwerk sind also eine höhere Geschwindigkeit, als der Brenner hat und ein möglichst konstanter Datenstrom. Man kann das Problem vermeiden, indem man auf der Platte eine Zwischenkopie anlegt. Um das Image einer CD abzuziehen, verwendet man das Programm dd:

Image mit dd
zwischenparken

```
dd if=/dev/cdrom of=/tmp/img.iso  
cdrecord -v dev=0,3,0 speed=2 /tmp/img.iso
```

1.21.2 RW-Medien

Bei den Kommandos zum Brennen unterscheiden sich RW-Medien nicht von den normalen Medien. Die Besonderheit besteht darin, dass man sie löschen und wiederverwenden kann. Das Löschen kann beim Brennen erfolgen. Dazu gibt man dem cdrecord die Option blank=fast mit. Weitere Optionen für blank sind:

blank=	Wirkung
fast	minimales Löschen: PMA, TOC und pregap werden gelöscht.
track	löscht nur einen Track (bei Multisession)
all	komplette CD-RW löschen. Das wird etwas dauern.

Tabelle 1.35 Optionen zu blank

1.21.3 Multisession

Normalerweise wird eine CD an einem Stück gebrannt. Aber gerade bei der Datensicherung einer Workstation ist das Datenaufkommen oft so gering, dass man gern mehrere Sicherungen auf eine CD schreiben möchte. Dazu werden mehrere Tracks geschrieben. Die Trackposition wird mkisofs

mit dem Parameter `-C` mitgeteilt. Sie besteht aus zwei durch ein Komma getrennte Zahlen. Die eine gibt die Anfangsposition des zuletzt gebrannten Tracks an, und die zweite beschreibt die Position, an der die neue Session beginnen soll. Man kann dieses Zahlenpaar mit `cdrecord -msinfo` bestimmen. Beim Brennen wird `cdrecord` mit der Option `-multi` mitgeteilt, dass mit Multisession gearbeitet wird.

Der erste Track Der erste Track unterscheidet sich nur in der Option `-multi` von einem normalen Brennen. Damit wird vermieden, dass die CD abgeschlossen wird.

```
mkisofs -J -R -o ../image.iso .
cdrecord -v speed=2 dev=0,3,0 -multi ../image.iso
```

Die Folgetracks Im nächsten Schritt soll eine beliebige weitere Sitzung gebrannt werden. In der Variablen `TRACKPOS` werden die Daten für die Position auf der CD gespeichert. Den Wert der Variablen ermittelt man mit Hilfe des `cdrecord` von der CD im Laufwerk. `TRACKPOS` wird anschließend von `mkisofs` verwendet, um die Option `-C` zu besetzen. Die Option `-M` ermittelt von der eingelegten CD den vorherigen Track. Als Parameter können die SCSI-Informationen, wie sie bei `cdrecord` vorliegen, verwendet werden, oder man gibt den Pfad des Device an.

```
TRACKPOS=`cdrecord -msinfo dev=0,3,0`
mkisofs -J -R -f -o ../image.iso -C $TRACKPOS -M 0,3,0 .
cdrecord -v speed=2 dev=0,3,0 -eject -multi ../image.iso
```

Um eine Multisession-CD abzuschließen, wird beim `cdrecord` die Option `-multi` weggelassen.

Ein vollständiges Beispielskript für das Sichern mit Hilfe eines CD-Brenners findet sich im Abschnitt 2.8.8 (siehe S. 202).

1.21.4 Audio-CDs

Auch wenn UNIX oft nur mit den dicken Servern in Verbindung gebracht wird, kann man es gut im Multimediabereich einsetzen. Dort waren beispielsweise Systeme von Silicon Graphics sogar Vorreiter. Auch heute noch werden UNIX-Workstations bei der Entwicklung von Spielfilmen eingesetzt. So kann ein UNIX-System heutzutage selbstverständlich auch Audio CDs abspielen und auch brennen.

Eine Audio-CD wird aus mehreren Tracks erstellt, die jeweils aus einer Musikdatei gespeist werden. Dabei gibt es zwei gängige Formate. Das eine sind die WAV-Dateien, deren Format von Microsoft definiert wurden, und

das andere sind die AU-Dateien, die aus dem Hause Sun stammen. Hat man seine Musik in diesen Formaten vorliegen, kann man mit ihnen und der Hilfe von `cdrecord` normale Audio-CDs herstellen, die man auch in der heimischen Stereoanlage hören kann.

```
cdrecord -dev 0,3,0 -speed=2 -pad -audio *.wav
```

Die Option `-pad` ist notwendig, wenn Sie Direktaufnahmen von Musikcassetten oder alten Schallplatten auf CD sichern wollen. Die Länge der Stücke passt selten genau auf die Blocklänge von CDs.

Braucht man einen Titel von einer bereits gebrannten CD als WAV-Datei, kann man ihn mit `cdparanoia` herunterlesen. Da `cdparanoia` direkt auf das Device `/dev/cdrom` zugreift, muss hier ein Link auf das CD-Laufwerk liegen. Selbstverständlich muss man auch Leserechte auf das Device haben.

**Auslesen von
Audio-CDs**

```
cdparanoia 5 mysong.wav
```

Damit wird der Track 5 von der CD gelesen und als Datei `mysong.wav` gespeichert. Mit dem Parameter `-B` liest `cdparanoia` alle Tracks der CD aus und gibt den einzelnen Dateien fortlaufende Namen.

Quelle: <http://www.xiph.org/paranoia/>

MP3-CDs

Audiodateien sind sehr groß. Mit MP3 ist es möglich, diese Dateien auf ein Zehntel zu reduzieren. Der Klangverlust ist so gering, dass er nicht ins Gewicht fällt, insbesondere wenn man die Musik unterwegs im Auto oder im Zug hört.

CDs für MP3-Player werden nach ISO-9660 gebrannt, also wie gewöhnliche CDs für Daten. Die MP3-Musikdateien werden als gewöhnliche Dateien abgestellt. Die Erweiterungen Joliet und Rockridge stören nicht. Die neueren MP3-Player beherrschen auch den Umgang mit Verzeichnissen, sodass man nicht alle Titel in das Wurzelverzeichnis stellen muss. Das kann wichtig sein, weil eine CD mit MP3 weit über 100 Musiktitel aufnehmen kann. Immerhin ist eine MP3-Datei im Schnitt zwischen 2 und 5 MByte groß.

Für das Konvertieren von WAV nach MP3 kann man das Programm `notlame` verwenden. Für die Gewinnung von WAV aus MP3 gibt es das Programm `mpg123`.

MP3-Dateien auspacken Will man MP3-Dateien zum Brennen konventioneller Audio-CDs verwenden, muss man zunächst WAV- oder AU-Dateien generieren. Das kann das Programm mpg123:

```
mpg123 -w output.wav input.mp3
```

Mit diesen WAV-Dateien kann man dann, wie oben gesagt wurde, eine normale CD für die Verwendung in gewöhnlichen CD-Spielern herstellen. Natürlich wird die Qualität nicht so gut sein wie eine normale CD, da MP3 ja nun Kompromisse im Klang eingeht.

Generieren von MP3-Dateien Für das Erzeugen von MP3-Dateien gibt es mehrere Tools. Einen guten Ruf hat das Produkt LAME. Da dafür Lizenzen fällig werden, ist dieses Produkt vor allem im professionellen Bereich zu finden. Für den Freizeitbereich gibt es Alternativen, wie beispielsweise notlame:

```
notlame --tt "Is That All" --ta "Headache" input.wav output.mp3
```

Zu notlame gibt es zwar keine Manpage, aber mit der Option --help erhält man recht ausgiebige Informationen. Nähere Informationen finden sich im Internet unter der folgenden Adresse:

<http://www.sulaco.org/mp3>

1.22 Notebooks

UNIX kann man auch auf Notebooks benutzen. Tragbare UNIX-Workstations sind keine neue Idee. So gab es schon vor zehn Jahren das SPARC-book von SUN. Allerdings waren diese Geräte extrem teuer und wurden dementsprechend als Demomaschinen im Vertrieb oder für Schulungen und auch nur da, wo es zu aufwändig gewesen wäre, eine Workstation mit Monitor zu transportieren. Der Preis der Geräte war so immens, dass sie keine weite Verbreitung fanden.³²

Peripherie und Kompatibilität Besonders die PC-UNIX-Systeme Linux und FreeBSD ermöglichen es nun, UNIX auch auf preisgünstigen Notebooks zu benutzen. Notebooks bringen allerdings aus mehreren Gründen zusätzliche Probleme mit sich, die eine normale Workstation nicht hat. Zunächst kann die Peripherie nicht so einfach ausgetauscht werden, wenn man entdeckt, dass es dafür keinen Treiber gibt. Das ist insbesondere ein Problem, wenn sich herausstellt, dass für den verwendeten Grafikkontroller kein Treiber für das X Window System existiert. Viele Hersteller halten es für überflüssig, den Kunden über den Grafikchip zu informieren. Wenn es dann keinen passenden Treiber

³² vgl. Kienle, Michael/Jaeschke, Gerhard: Reisebegleiter. iX 7/1993, S. 26–34.

gibt, kann man auch nicht eine andere Grafikkarte kaufen und einbauen. Die PCMCIA-Karten können, anders als die gewohnten Steckkarten, im laufenden Betrieb gewechselt werden. Dadurch wird dem System ein fliegender Wechsel in der Konfiguration abverlangt. Zu guter Letzt wünscht sich der Besitzer die Möglichkeit, den Akku zu überwachen, zumindest aber, dass alle Möglichkeiten genutzt werden, Energie zu sparen, wenn er unterwegs mit dem Gerät arbeiten will. Informationen zu UNIX auf Notebooks finden Sie unter:

<http://www.mobilix.org>

1.22.1 PCMCIA

Notebooks besitzen als Erweiterungsmöglichkeiten ein bis zwei Schächte, in die man so genannte PC-Cards oder auch PCMCIA-Karten einstecken kann. Die Karten haben die Größe einer Kreditkarte und können im laufenden Betrieb gewechselt werden. Sowohl bei FreeBSD als auch bei Linux arbeitet auf einem Notebook ein Prozess im Hintergrund, der die entsprechenden Slots überwacht. Der Dämon besitzt eine Konfigurationsdatei, in der die bekanntesten Karten und ihre Erkennungsmerkmale abgestellt sind.

Unter FreeBSD wird der Dämon `pccardd` eingesetzt, um die PCMCIA-Slots zu überwachen. Als Information für die verschiedenen Karten dient die Datei `pccard.conf` im Verzeichnis `/etc/default`. Im Verzeichnis `/etc` befindet sich eine weitere Datei `pccard.conf`, die eingebunden wird. Die Einträge enthalten die Zeichenketten, mit denen sich eine Karte beim System zu erkennen gibt, und enthält die Optionen und die zu startenden Treiber. In dem folgenden Ausschnitt der Datei `pccard.conf` ist ein Eintrag für eine Netzwerkkarte zu sehen:

`pccardd` unter
FreeBSD

```
# "Ethernet Adapter" "E2000 PCMCIA Ethernet"
card "Ethernet Adapter" "E2000 PCMCIA Ethernet"
    config auto "ed" ?
    insert /etc/pccard_ether $device start
    remove /etc/pccard_ether $device stop
```

Interessant ist hier vor allem die erste Zeile. Wenn eine Karte sich mit diesen beiden Zeichenketten meldet, dann wird beim Einschieben der Befehl gestartet, der hinter dem Stichwort `insert` steht. Es kann sein, dass die erworbene Karte zwar nicht erkannt wird, aber durchaus kompatibel zu einer anderen Karte in der `pccard.conf` ist. Unter welchem Namen die Karte sich gemeldet hat und vom `pccardd` abgewiesen wur-

de, sieht man auf der Konsole oder in jedem Fall in der Protokolldatei `/var/log/messages`.

cardmgr und Linux Der PC-Card-Manager `cardmgr` wird unter Linux beim Booten gestartet und liest die Datei `/etc/pcmcia/config`. Darin werden alle bekannten PCMCIA-Karten aufgeführt. Auch hier werden vergleichbare Mechanismen benutzt, um PC-Cards zu erkennen und zu starten.

1.22.2 Problematische Peripherie

Nicht alles, was in einem Notebook eingebaut ist, lässt sich problemlos unter UNIX betreiben. Man findet mit Hilfe der gängigen Suchmaschinen aber leicht Informationen darüber, welche Systeme mit welchen Notebooks laufen. Sobald Sie das Notebook mit einer grafischen Oberfläche verwenden wollen, empfiehlt sich dringend ein Blick ins Internet. Schwierig können auch die eingebauten Netzwerkkarten werden. Unter Linux ist das meist weniger kritisch als unter FreeBSD. Bei den eingebauten Modems neuerer Generation handelt es sich meist um solche, die nur unter MS Windows betrieben werden können. Das ist aber kein Drama, da man ja ein Modem leicht als PCMCIA-Karte bekommen kann.

Internet Für nahezu jedes Notebook findet man im Internet Seiten, die beschreiben, wie man Linux auf diesem Gerät installieren kann. Dort findet man normalerweise auch Links auf die Treiber und Hinweise darauf, wenn die Peripherie nicht ansprechbar ist. Man findet die Seiten leicht über jede beliebige Suchmaschine mit dem Notebook-Typ und den Wörtern »Linux« und »installation« als Stichworten.

1.22.3 Software für den Akku

Sowohl Linux als auch FreeBSD unterstützen das Advanced Power Management (APM) eines Notebooks. Dabei wird im BIOS der Status der Stromversorgung abgefragt. Der Dämon `apmd` überwacht den Stand der Batterien. Es gibt Hinweise darauf, dass nicht alle Notebooks die Standards des APM sauber einhalten. Im Zweifel sind die Manpages zu konsultieren oder die unten angegebenen Webseiten. Weite Informationsquellen sind die HOWTOs »Battery-Powered« und »Laptop«. Leider gibt es von beiden bislang keine deutsche Übersetzung.

Mit installiertem `apm` wird ein `shutdown -h`, also ein halt, letztlich zum Ausschalten der Maschine führen.



Abbildung 1.4 Akkuüberwachung mit xapm

Unter X läuft das Programm `xapm`. In der linken Hälfte seines Fensters wird eine Zeichenfolge angezeigt. Der Buchstabe B zeigt an, dass das Gerät auf Batterie läuft; ein P steht für die Spannungsversorgung durch das Netzteil. Im Batteriemodus steht dann der Prozentsatz der Ladung oder die Zeit, die der Akku das Gerät noch mit Spannung versorgen kann. In Abbildung 1.4 zeigt das Programm eine Restdauer von einer Stunde und sechs Minuten an. Durch einen Mausklick kann man zwischen den Anzeigen umschalten. In der rechten Hälfte ist ein Balken zu sehen, der den Füllgrad der Akkus grafisch anzeigt.

Zum Thema Advanced Power Management und dessen Entwicklung unter Linux finden sich weitere Informationen auf der Seite:

<http://www.cs.utexas.edu/users/kharker/linux-laptop/apm.html>

Die Homepage des `apmd` ist unter folgender URL zu finden:

<http://www.cut.de/bkr/linux/apmd/apmd.html>

Strom sparen ohne APM

Sollte das Notebook APM nicht unterstützen, kann man immerhin versuchen, die Festplattentätigkeit auf ein Minimum zu reduzieren. Damit ist immerhin einer der wichtigsten Verbraucher neben der CPU zu beeinflussen.

Platten abschalten

Mit dem Programm `hdparm` und der Option `-S` lässt sich die Zeit einstellen, die nach der letzten Plattenaktivität vergehen soll, bevor der Festplattenmotor abgestellt wird. Dadurch lässt sich einiges an Energie sparen. Der Wert hinter `-S` hat folgende Bedeutung:

- ▶ **1–240** Dieser Wert, multipliziert mit 5 Sekunden, ist die Zeit, nach der die Platte stoppt.
- ▶ **241–251** Dieser Wert minus 240, multipliziert mit 30 Minuten, ist die Zeit, nach der die Platte stoppt.
- ▶ **252** Timeout von 21 Minuten.

Beispiel:

```
hdparm -S 120 /dev/hda
hdparm -S 242 /dev/hda
hdparm -S 0 /dev/hda
```

Im ersten Fall wird die Platte nach zehn Minuten, im zweiten nach einer Stunde Inaktivität abgeschaltet, und im dritten Fall wird gar keine Plattenabschaltung durchgeführt.

crontab In der Datei **/etc/crontab** sollte man nach Einträgen suchen, die häufig durchlaufen werden. Wird beispielsweise `atrun` jede Minute gestartet, ist es eher unwahrscheinlich, dass die Platte jemals zur Ruhe kommt. In den meisten Fällen ist es eine praktikable Lösung, wenn die `at`-Jobs nur jede Stunde einmal laufen (zum Thema `at` siehe Seite 111).

Weitere Hinweise finden sich im Linux-HOWTO »Battery Powered«.

Administration

- Administration
 - Die Arbeitsumgebung des Administrators
 - Administrationstools
 - Start des Systems
 - Herunterfahren: shutdown
 - Benutzerverwaltung
 - Hardwarezugriff unter UNIX: /dev
 - Festplatten
 - Datensicherung
 - Software installieren
 - Druckeradministration
 - Terminals
 - Anschluss eines Modems
 - Tuning
 - Informationen sammeln
 - Der Kernel
-

Administration

Der Administrator einer UNIX-Maschine ist nicht die Nummer 1, sondern hat die Benutzernummer 0. Er hat die unbeschränkte Macht, alles zu reparieren, aber auch alles zu zerstören.

Traditionsgemäß hat der Superuser den Namen root. Auch wenn dieser Name geändert werden kann, sollte man es der Einfachheit dabei belassen. Weil seine Macht ein großes Risiko darstellt, wird der Systemadministrator neben root ein gewöhnliches Benutzerkonto haben und sich nur dann als root einloggen, wenn es wirklich erforderlich ist.

-
- Die Arbeitsumgebung des Administrators
 - Administrationstools
 - Start des Systems
 - Bootprompt
 - Bootkonfiguration: lilo
 - Durchlaufen der Runlevel
 - BSD: /etc/rc
 - System V: init.d
 - Konfigurationsdateien
 - Herunterfahren: shutdown
 - Alles bereit zum Untergang?
 - Wechsel in den Single-User-Modus
 - Benutzerverwaltung
 - Passwortverwaltung unter UNIX
 - Die Benutzerdatei /etc/passwd
 - Verborgene Passwörter: shadow
 - Benutzerpflege automatisieren
 - Grundeinstellungen: /etc/profile
 - Verzeichnisprototyp: /etc/skel
 - Gruppenverwaltung
 - Benutzerüberwachung
 - Kurzfristiger Benutzerwechsel: su
 - Administrationsaufgaben starten: sudo
 - Pseudob Benutzer zum Shutdown
 - Hardwarezugriff unter UNIX: /dev
 - Aufgaben eines Treibers
 - Gerätedateien

- Umgang mit Gerätedateien
- Gerätenamen
- Festplatten
 - SCSI-Platten
 - IDE-Platten
 - Inbetriebnahme
 - RAID-Systeme
 - Partitionieren
 - Dateisystem erstellen
 - Swapping
 - Einbinden eines Dateisystems: mount
 - Konsistenz der Dateisysteme
 - Journal-Dateisysteme
 - Belegungslisten: df und du
 - Zuteilung des Plattenplatzes: quota
 - Maximalwerte
- Datensicherung
 - Vorüberlegungen
 - Das Bandlaufwerk
 - dump
 - tar (tape archiver)
 - cpio
 - Medien kopieren: dd
 - Andere Sicherungstools: AMANDA
 - Beispiel für eine Sicherung auf CD-RW
- Software installieren
 - make als Installationswerkzeug
 - Solaris Packages
 - HP-UX: SD-UX
 - Red Hat Package Manager
- Druckeradministration
 - Übersicht
 - BSD-Unix: lpd, lpr, lpq und lprm
 - Linux-PC als Druckserver
 - System V: lpsched, lp, lpstat und cancel
 - LPRng
 - CUPS - Common UNIX Printing System
- Terminals
 - Konfiguration der Terminals
 - Die Terminalvariable TERM
 - termcap
 - terminfo
 - Wenn das Terminal durcheinander ist
- Anschluss eines Modems

- Tuning
 - Optimierung des Dateisystems
 - Wissen, wo der Schuh drückt
 - Informationen sammeln
 - Versionsinformationen: uname
 - Der syslog-Dämon und die messages-Datei
 - Umgang mit großen Protokolldateien
 - Briefe aus dem Nirvana
 - Bootzeitpunkt und Systemlast: uptime
 - Prozessbeobachter
 - Nicht immer mit Tötungsabsicht: kill
 - Offene Dateien
 - Programmzusammenbrüche (Core Dump)
 - Systemabsturz (Kernel-Panic)
 - Der Kernel
 - Dynamische Bibliotheken
 - Module
-

Die Arbeitsumgebung des Administrators

Die Arbeitsumgebung des Administrators ist durch seine besondere Rolle gekennzeichnet. Er ist das besondere Ziel von Angriffen, durch seine besonderen Rechte können seine Fehler ungebremsst Schaden anrichten, und er muss auch noch arbeiten können, wenn die Maschine nicht mehr ganz in Ordnung ist.

Eine normale UNIX-Maschine hat ihre Daten auf mehrere Platten verteilt. Jede dieser Platten kann ausfallen. Bei der Verteilung des Systems versucht man nun, die Dateien, die man braucht, um das System noch halbwegs starten zu können, auf einer Platte zusammenzustellen. Dazu zählen der Kern des Systems, die Startskripten, die Shell, ein Editor und die wichtigsten Werkzeuge, um Platten reparieren zu können und um eine Notsicherung durchzuführen. Die Anmeldung als root muss mit dieser Minimalumgebung möglich sein. Darum sind die Programme, die root vorfindet, klein und manchmal wenig komfortabel.

Unter Linux gibt es sogar Projekte, ein von Diskette bootfähiges System zu erstellen, das genügend Werkzeuge an Bord hat, um ein vollständiges System zu starten. Mit solchen Werkzeugen kann man in der Not eine Linux-Maschine wieder lauffähig machen. Man kann solch ein System aber auch zur Netzüberwachung oder als Router oder Druckserver einsetzen. Da diese Systeme ohne Festplatte und CD auskommen, können sie auch in Umgebungen eingesetzt werden, in denen der Betrieb eines Computers sonst eher schwierig ist. Eine solche Distribution finden Sie auf der folgenden Webseite:

<http://www.toms.net/rb/>

Die Environment-Variable PATH sollte für root keinen Punkt enthalten. Das Fehlen des Punktes hat zur Folge, dass root keine Programme des aktuellen Verzeichnisses direkt aufrufen kann. Will man dies dennoch tun, muss man den Pfad voranstellen. Dazu reicht es aus, Punkt und Schrägstrich vor den Dateinamen zu stellen. Die Unbequemlichkeit hat ihren Grund darin, dass root nicht aus Versehen Skripten oder Programme im aktuellen Verzeichnis aufrufen soll, die dann immerhin mit Administratorrechten ausgeführt werden.

Auf einigen Systemen erhält der Benutzer root beim Aufruf von `rm` automatisch die Option `-i` aufgedrückt. Das heißt, dass root jede Datei, die er löscht, bestätigen muss. Wer einmal schlechte Erfahrungen mit versehentlichem Löschen gemacht hat, wird sich diese Rückfrage gern selbst einrichten. Es geht ganz leicht mit einem alias:

```
alias rm='rm -i'
```

Administrationstools

Auf beinahe jeder UNIX-Plattform gibt es ein anderes Administrationstool. Leider hat sich hier kein Standard herausgebildet. Vielleicht hängt es damit zusammen, dass diese Tools auch Hardwarekonfigurationen mitverwalten. Immerhin sind sie recht ähnlich in der Handhabung. Menügesteuert handelt man sich zur anstehenden Aufgabe und bekommt dann eine Seite mit einer Eingabemaske präsentiert, die alle Parameter der zu erledigenden Aufgabe abfragt. Anschließend ruft das Tool die Standardwerkzeuge auf oder nimmt die entsprechenden Einträge in den Systemdateien vor.

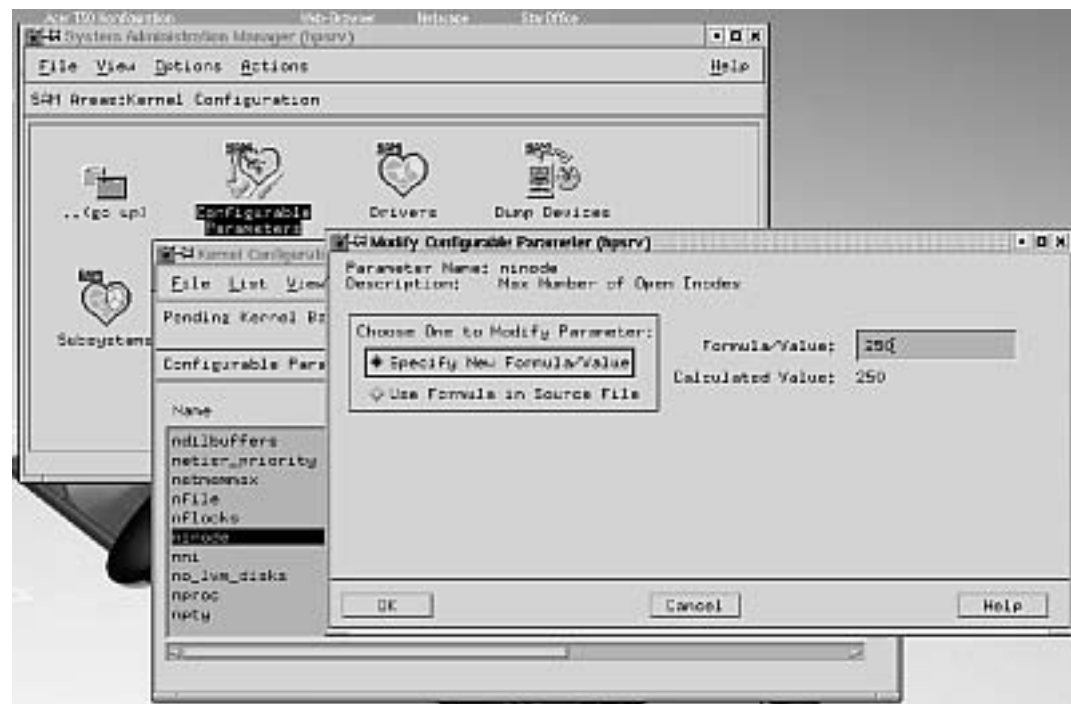
[Administrationstools auf verschiedenen Systemen]L|L UNIX-Version & Name des Administrationstools

AIX	smit
HP-UX	sam
SCO	scoadmin bzw. sysadmsh
S.u.S.E.- Linux	yast
Red Hat Linux	linuxconfig
Solaris	admintool
FreeBSD	/stand/sysinstall
MacOS X	System Preferences

Es macht wenig Sinn, die einzelnen Programme hier im Detail zu beschreiben. Im Allgemeinen verwendet man sie, um Softwarepakete nachzuinstallieren, Benutzer zu verwalten, den Kernel anzupassen und grundlegende Dinge einzustellen, wie beispielsweise die TCP/IP-Nummer (siehe S. ipnr). Die Programme sind normalerweise intuitiv zu verstehen, allerdings manchmal etwas unbequem zu benutzen. Natürlich können alle diese Arbeiten auch ohne das Administrationstool durchgeführt werden, wie in den folgenden Abschnitten zu sehen ist. Der Vorteil der Tools ist, dass die Änderungen systemgemäß vorgenommen werden. Der Leistungsumfang der Programme ist durchaus unterschiedlich. So wird in einigen Fällen beim Neuanlegen eines Benutzers für diesen lediglich ein Eintrag in der Benutzerdatei vorgenommen, während andere Tools auch dessen Heimatverzeichnis anlegen und die komplette Umgebung erstellen. Ein paar Besonderheiten einzelner Tools seien hier noch genannt.

Das AIX-Tool `smit` zeigt als Besonderheit die Systembefehle an, die es für jeden Schritt verwendet. Dadurch lernt man die wichtigen Tools der Maschine kennen und weiß definitiv, was im Hintergrund passiert.

`sam` arbeitet als Konsolentool, das allerdings eine gute Terminalemulation braucht, um bedienbar zu bleiben. Wenn man kein Original-HP-Terminal besitzt oder gar auf eine Terminalemulation angewiesen ist, wird die Bedienung von `sam` schwierig. Zum Glück kann `sam` aber auch als X-Anwendung gestartet werden. Es erkennt anhand der Variablen `DISPLAY`, ob ein X-Server zur Verfügung steht. Damit kann dann die Administration auch von einer fremden Maschine aus durchgeführt werden (siehe S. display).



yast (wie auch dessen Nachfolger yast2) führt eine Konfiguration anhand einer speziellen Konfigurationsdatei `/etc/rc.config` durch. Aus den Informationen dieser Datei werden dann die Systemdateien generiert. Man findet darum in den Kommentaren dieser Dateien oft den Hinweis, man möge sie nicht direkt verändern, sondern über die Datei `rc.config` gehen.

Der Vorteil dieser Tools ist, dass man nicht so viele Fehler machen kann und dass Querabhängigkeiten durch das Tool überwacht werden. Nicht immer ist das Tool für jede Situation flexibel genug. Aber es gibt auch immer einen Weg an den Tools vorbei.

« [Die Arbeitsumgebung des Administrators](#) | [Administration](#) | [Start des Systems](#) »

Start des Systems

Der Start einer UNIX-Maschine ist normalerweise einem durchschnittlich intelligenten Menschen binnen Sekunden erklärt: Man drückt auf den Hauptschalter und wartet auf die Aufforderung zum Einloggen.

Vom Start zum Login machen einige Maschinen an zwei markanten Punkten eine kurze Rast. Die erste Position ist der Bootprompt. Hier hat man die Möglichkeit, der Maschine Bootparameter mitzugeben oder das Medium zu nennen, von dem gebootet werden soll. In den meisten Fällen wird der Bootvorgang nach einigen Sekunden von selbst fortgesetzt. Ansonsten muss man Return oder das Wort boot, gefolgt von Return eingeben.

Der zweite Halt erfolgt im Single-User-Modus. Hier wird man entweder nach dem root-Kennwort gefragt oder befindet sich direkt in der root-Shell. Geht es hier nicht von selbst weiter, reicht ein Ausloggen durch `ctrl-D`, `exit` oder `logout`. Der Single User Modus hat bei der Wartung einer UNIX-Maschine besondere Bedeutung. Hier kann beispielsweise die Konsistenz eingebundener Platten geprüft und gegebenenfalls wieder hergestellt werden. Sollte der Start der Maschine trotz eines Ausloggens nicht weitergehen, muss man die Meldungen sehr genau lesen und nach den Problemursachen suchen.

Eine bootende UNIX-Maschine erschlägt den Benutzer mit Meldungen. Die Hardware wird aufgelistet und initialisiert, die verschiedenen Betriebssystembestandteile melden, dass sie vorhanden und einsatzbereit sind. Während diese Meldungen am Anfang verwirrend wirken, bekommt man bei häufigerer Beobachtung ein Gefühl dafür, welche Systembestandteile wann starten und welche Zusammenhänge existieren. Wenn es dann einmal ein Fehlverhalten gibt, kann man es leicht erkennen und vielleicht sogar die Ursache analysieren.

Unterabschnitte

- [Bootprompt](#)
 - [Bootkonfiguration: lilo](#)
 - [Durchlaufen der Runlevel](#)
 - [BSD: /etc/rc](#)
 - [System V: init.d](#)
 - [Konfigurationsdateien](#)
-

Bootprompt

Der Bootvorgang einer Maschine beginnt damit, dass im ROM ein kleines Programm steht, das die Maschine anweist, den ersten Block der Platte zu laden und auszuführen. In diesem ersten Block steht ein etwas umfangreicheres Programm, das bereits in der Lage ist, ein UNIX-Dateisystem zu lesen und so den eigentlichen Betriebssystemkern in den Speicher zu laden und durchzustarten.

Moderne Computer sind weitgehend konfigurierbar. Eine Maschine soll von der Platte, von der CD-ROM, aus dem Netz oder sogar vom Bandlaufwerk booten. Diese Information wird in einem RAM abgelegt, das durch einen Akku oder eine Batterie gesichert wird. Bei Suns Workstations nennen sich die entsprechenden Bausteine NVRAM. Sie enthalten den Speicher inklusive der Batterie. Wenn ihre Batterie leer ist, vergessen die Maschinen alles, bis hin zu ihrer Ethernet- und Rechnernummer. Erst recht wissen sie nicht mehr, von welchem Medium sie booten sollen. Stellt man erste Probleme beim Booten fest, sollte man genaue Aufzeichnungen über alle Informationen haben, die bei einem Ausfall der Batterie wichtig sind. Für weitere Informationen zu diesem Thema gibt es eine FAQFrequently Asked Questions:

<http://www.squirrel.com/squirrel/sun-nvram-hostid.faq.html>

Bootkonfiguration: lilo

Während eine reine UNIX-Maschine darauf ausgelegt ist, einen UNIX-Kernel zu booten, muss auf einem PC ein Bootmanager eingerichtet werden, der nach dem Start des BIOS Das ist das Boot-ROM eines PCs. vom Master Boot Record einer Platte geladen wird und von dort den Kernel aus dem Dateisystem des UNIX startet. Auf der anderen Seite müssen diese Bootprogramme auch so flexibel sein, andere Betriebssysteme auf dem Rechner zu starten oder sich von anderen Bootmanagern starten zu lassen. Eines der ausgereiftesten Programme dieser Art ist sicher das Programm `lilo` (linux loader) für Linux. `lilo` wird durch die Datei `/etc/lilo.conf` gesteuert:

```
image = /boot/vmlinuz.suse
  root = /dev/hda5
  label = Linux
```

```
other = /dev/hda1
  label = windows
  table = /dev/hda
```

In der Datei werden die einzelnen Partitionen (siehe S.partition) aufgezählt, die durch Lilo gestartet werden sollen. Der Eintrag für eine Linux-Partition wird durch das Schlüsselwort `image` eingeleitet und zeigt auf den Pfad des Kernels. Andere Betriebssysteme, wie hier MS-Windows, werden durch das Schlüsselwort `other` eingeleitet. Mehrfachnennungen sind möglich. Der Name, unter dem das gewählte Betriebssystem startet, steht hinter dem Schlüsselwort `label`. Auch Bootparameter, die für das Starten von Linux bei bestimmter Hardware benötigt werden, können hier hinterlegt werden.

Nach der Erstellung der `lilo.conf` wird einmal der Befehl `lilo` aufgerufen, um die Einträge zu übernehmen. Dabei wird eine Liste der Optionen angezeigt. Die mit einem Stern versehene Partition wird dabei automatisch gestartet.

Lilo kann man von MS-DOS oder MS-Windows aus durch den Aufruf von `FDISK /MBR` wieder entfernen. Das Programm `fdisk`, das bei MS Windows mitgeliefert wird, hat manchmal Probleme damit, Linux-Partitionen zu entfernen. Wenn man als Linux-Anwender einem MS Windowsbenutzer eine Festplatte überlässt, tut man ein gutes Werk, wenn man vorher alle Partitionen entfernt.

Für den PC gibt es diverse andere Bootmanager. Aber selbst wenn ein solcher eingesetzt wird, wird Lilo gebraucht. Lilo wird dann auf der Bootpartition selbst installiert, um dem Bootmanager vorzutäuschen, dass sich Linux genau wie MS-Windows oder MS-DOS verhält.

« [Bootprompt](#) | [Start des Systems](#) | [Durchlaufen der Runlevel](#) »

Durchlaufen der Runlevel

Beim Bootprozess von UNIX spricht man von Runleveln, die durchlaufen bzw. erreicht werden. Die bekanntesten sind der Single-User-Modus mit dem Runlevel 1 und der Multiuser-Betrieb mit Runlevel 2. Für den Wechsel von einem Runlevel zu einem anderen kann man durch den Aufruf des Kommandos `init` mit dem Runlevel als Parameter sorgen. Es versteht sich von selbst, dass nur der Administrator dieses Kommando geben kann.

Der allererste Prozess beim Booten der Maschine ist der Kernel (siehe S. kernel). Als Kernel bezeichnet man das zentrale Betriebssystem selbst. Der Kernel findet sich an wohlbekannter Stelle im Verzeichnisbaum. Zu Anfang lag der Kernel im Wurzelverzeichnis unter dem Namen `unix`, auf einigen Systemen hieß er auch `vmunix`. Inzwischen liegt er meist im Verzeichnis `/boot`. Er hat die Prozessnummer 0. Da das Betriebssystem schlecht gleichzeitig ein Prozess sein kann, startet es als Erstes den Prozess `init`, der entsprechend die Prozessnummer 1 erhält. Damit ist `init` der Urahn aller Prozesse. Dieser Prozess liest bei System V die Datei `/etc/inittab` und bei BSD-Systemen die `rc`-Dateien. Die Datei `inittab` enthält typischerweise die Informationen über das Durchlaufen der Runlevel.

[Die UNIX-Runlevel]C|L Runlevel & Zustand
 0 & Maschine steht (halt)
 S & Single-User
 1 & Multiuser ohne Netzwerk
 2 & Multiuser mit Netzwerk
 3 & Multiuser mit Netzwerk und grafischem Login
 6 & Reboot

In Tabelle finden Sie die Runlevel. Leider sind sie nicht ganz einheitlich. Auf manchen Systemen ist die Nummerierung etwas anders. So ist bei Linux der grafische Login der Runlevel 5. Dafür wird auf vielen Systemen der Single-User-Modus mit `init 1` erreicht. Bei diesen Systemen gibt es keinen Multiuser-Betrieb ohne Netzwerk. Konkrete Informationen für Ihr System finden Sie auf der Manpage von `init`.

Im Single-User-Modus läuft nur die Konsole. Das ist entweder das Terminal, das über die erste serielle Schnittstelle angeschlossen ist, oder der im Gerät integrierte Arbeitsplatz. In diesem Zustand kann sich nur eine Person anmelden. Im Hintergrund läuft bereits der Swapdienst (siehe S. swap); die wichtigsten Geräte können angesprochen werden.

Wie schon gesagt wurde, kann sich in diesem Modus nur ein Benutzer anmelden. Alle Terminals sind noch nicht aktiv. Auch das Netzwerk ist nicht ansprechbar. Da keine weiteren Benutzer stören können, ist dies der ideale Zustand, um Wartungsarbeiten durchzuführen, insbesondere wenn es darum geht, Festplatten zu prüfen.

Hier erfolgt das Starten der `getty`-Prozesse, die die Terminals bedienen. Es können sich also bei Erreichen dieses Levels alle Benutzer anmelden. Nicht jede UNIX-Version unterstützt diesen Zustand. Normalerweise wird mit dem Multiuser-Modus auch gleich das Netzwerk mitgestartet.

Neben der Möglichkeit des Einloggens per Terminal werden TCP/IP und alle zugehörigen Dienste gestartet. Dies ist der typische Runlevel für eine Servermaschine.

Die grafische Oberfläche ist in diesem Zustand nicht gestartet, es gibt also keinen grafischen Login. Man kann aber, sofern die Voraussetzungen dafür da sind, jederzeit mit dem Befehl `startx` eine grafische Oberfläche in Betrieb nehmen.

Hier wird der grafische Login erreicht. Ein Benutzer arbeitet also von vornherein unter einer grafischen Oberfläche. Workstations arbeiten normalerweise in diesem Runlevel. Natürlich ist es auch möglich, sich parallel über ein Terminal anzumelden oder das Gerät als Server zu verwenden.

Mit `init 6` kann UNIX zum Reboot aufgefordert werden.

Obwohl der Runlevel 0 (`halt`) die niedrigste Nummer hat, ist er der letzte, der erreicht wird. Die Maschine befindet sich im heruntergefahrenen Zustand, alle Prozesse sind beendet. Man kann den Computer ausschalten oder neu starten. Dieser Modus wird mit `init 0` oder mit dem Befehl `shutdown` erreicht.

« [Bootkonfiguration: lilo](#) | [Start des Systems](#) | [BSD: /etc/rc](#) »:

BSD: /etc/rc

Das Starten der Systemumgebung erfolgt unter BSD-UNIX durch das Ausführen der Datei /etc/rc. Diese Datei startet nacheinander die relevanten Startdateien, die sich typischerweise ebenfalls im Verzeichnis /etc befinden und mit rc. beginnen.

Zuvor werden die Dateien rc.conf in den Verzeichnissen etc/default und /etc gestartet. Sie besetzen Umgebungsvariablen mit Werten, die von den rc-Dateien verwendet werden, um die Dämonen Zum Begriff Dämon siehe S. daemon. zu konfigurieren. Das heißt, dass Änderungen an der Konfiguration, wie beispielsweise eine Änderung des Rechnernamens hier zu geschehen haben.

System V: init.d

Je nach System befinden sich im Verzeichnis /etc/init.d oder im Verzeichnis /sbin/init.d die Skripten für den Start der Systemprozesse, die beim Systemstart ausgeführt werden. Zum Thema Programmierung von Shellskripten siehe S. shellskript. Dabei handelt es sich ausnahmslos um Shellskripten, deren Kommandos mit einem normalen Editor bearbeitet werden können. So ist es möglich, hier mit Hilfe des Kommandos `grep` nach dem Start von bestimmten Prozessen zu suchen.

Um die Runlevel zu initialisieren, gibt es für jeden ein eigenes Verzeichnis `rcX.d`. Dabei steht X für den jeweiligen Runlevel. In jedem der Runlevelverzeichnisse befinden sich ausschließlich symbolische Links auf die eben betrachteten Startskripten. Es existiert immer dann ein symbolischer Link, wenn beim Übergang in den jeweiligen Runlevel das Skript ausgeführt werden soll. Dabei haben alle Links ein Präfix S oder K, gefolgt von zwei Ziffern. Das Skript mit dem Präfix S soll die Prozesse starten, und K soll sie beenden (K wie kill). Die Ziffern sorgen dafür, dass die Prozesse in der richtigen Reihenfolge gestartet werden. Beispielsweise heißen die Links in `rc2.d` für `apache`, den Webserver, `S20apache` und `K20apache`. Das erste Link wird beim Eintreten in den Runlevel 2 und der andere beim Verlassen des Runlevels aufgerufen. Die Nummer 20 bezeichnet die Reihenfolge. S19 wird beispielsweise vor S20apache aufgerufen. Muss also der neue Dienst vor dem Webserver starten, sollte S19 als Präfix verwendet werden. Braucht der neue Server allerdings die Anwesenheit von Apache, sollte der Name des Links mit S21 oder höher beginnen.

Man schreibt ein Startskript, das als Parameter die Wörter `start` und `stop` verarbeitet. Einige dieser Skripten verarbeiten auch das Kommando `restart`. Ein solches Skript, das man zunächst im Verzeichnis `init.d` ablegt, besteht aus einer großen `case`-Anweisung. Als Beispiel soll der Dienst `unfug` gestartet werden. Der Dienst `unfug` wird durch das Programm `unfugd` gestartet. Als Startskript erstellt man eine Datei namens `unfug` und stellt sie in das Verzeichnis `/sbin/init.d`:

```
#!/bin/sh
test -f /usr/bin/unfugd || exit 0
case "$1" in
    start)
        echo "Starte Unfug-Daemon"
        /usr/bin/unfugd
        ;;
    stop)
        echo "Stoppe Unfug-Daemon"
        kill `cat /var/run/unfugd.pid`
        ;;
    restart)
        echo "Restart Unfug"
        kill `cat /var/run/unfugd.pid`
        /usr/bin/unfugd
        ;;
    *)
        echo "usage: MATH
$0 start | stop | restart"
        exit 1
        ;;
esac$
```

In der ersten Zeile wird festgelegt, dass dieses Skript von der einfachen Bourne Shell gestartet wird.

Die zweite Zeile testet, ob es das Programm unfugd im Verzeichnis /usr/bin überhaupt gibt. Ist das nicht der Fall, endet das Skript hier mit der Fehlernummer 0. Es ist ja kein wirklicher Fehler aufgetreten. Es gibt nur einfach nichts zum Starten. Die case-Anweisung untersucht die Variable \$1, die für den ersten Parameter steht, und verzweigt auf start, stop, restart oder auf den Stern, mit dem alle anderen möglichen Werte gemeint sind.

Bei start gibt es eine kurze Meldung, und dann wird das Programm unfugd gestartet. Dämonen haben die Eigenart, von selbst in den Hintergrund zu gehen. Darum ist an dieser Stelle ein & nicht erforderlich. unfugd schreibt seine Prozess-ID in die Datei /var/run/unfugd.pid. Das macht es beim Herunterfahren einfacher, den Prozess wieder zu beenden. Beim stop wird nämlich genau dort das Argument für den Befehl kill ausgelesen. Beim restart wird unfugd erst beendet und dann neu gestartet. Bei vielen Dämonen hätte auch ein kill -SIGHUP ausgereicht. restart wird nicht für das Booten oder Herunterfahren gebraucht, ist aber sehr praktisch, wenn man den Dämon konfigurieren will. Der Stern sorgt einfach dafür, dass jemand, der das Skript falsch aufruft, auch einen Hinweis bekommt, was falsch ist.

Startskripten sollten sehr sorgfältig erstellt werden. Dabei ist ein Abbruch nicht einmal so dramatisch. Falls aber ein Startskript hängen bleibt oder in einer Endlosschleife verharret, bleibt der Rechner beim Booten stehen. In einer solchen Situation brauchen Sie eine Möglichkeit, den Computer von einem anderen Bootmedium zu starten. Anschließend binden Sie die Platte per mount (siehe S. mount) ein und können dann das Skript löschen oder sogar bearbeiten.

Ausführliche Informationen über das Schreiben von Shellskripten und die verwendeten Anweisungen folgen später (siehe S. shellskript). Will man ein Skript beim Systemstart hochfahren, muss man zunächst überlegen, in welchem Runlevel es gestartet werden soll, und dann mit den anderen Links vergleichen, zu welchem Zeitpunkt es gestartet werden muss.

Nachdem nun das Skript unfugd in init.d liegt, muss man es noch mit chmod 755 ausführbar machen. Zuletzt wird es an passender Stelle in ein rcX.d-Verzeichnis verlinkt. Der Unfug soll erst nach dem apache starten und vor diesem herunterfahren. Also verwendet man das Verzeichnis rc2.d und erzeugt einen Link mit S21 und einen mit K19:

```
cd rc2.d
ln -s ../unfug S21unfug
ln -s ../unfug K19unfug
```

Die Ausgabemeldungen des letzten Bootprozesses werden normalerweise in einer eigenen Datei protokolliert. Unter HP-UX ist das /etc/rc.log, unter Linux /var/log/boot.msg.

Konfigurationsdateien

Während ein Systemadministrator in früheren Jahren die rc-Skripten an die Gegebenheiten seines Systems anpasste, wird heute oft mit sehr allgemeinen rc-Skripten gearbeitet, die ihre Informationen aus Konfigurationsdateien beziehen.

Unter BSD finden Sie die Konfigurationsdaten in den bereits erwähnten Dateien rc.config im Verzeichnis /etc/default und /etc.

HP-UX legt diese Konfigurationsdateien im Verzeichnis /etc/rc.config.d ab. Hier befinden sich dann viele kleine Dateien, die manchmal nur eine Information enthalten, die in den rc-Skripten direkt verwendet werden.

Die SuSE-Distribution verwendet eine Datei namens /etc/rc.config, über die Umgebungsvariablen verändert werden, die von den rc-Skripten gelesen werden. Diese Datei wird vom Administrationstool `yast` gepflegt, kann aber auch mit einem normalen Editor bearbeitet werden. Damit von Hand eingetragene Änderungen sofort übernommen werden, muss man anschließend `SuSEconfig` aufrufen. Bei Änderungen durch `yast` erledigt das Tool diese Aufgabe automatisch.

Ab der Version 8.0 hat SuSE die bisher in der Datei rc.config gesammelten Konfigurationen im Verzeichnis /etc/sysconfig auf mehrere Dateien und Unterverzeichnisse verteilt.

Herunterfahren: shutdown

Der Befehl zum Herunterfahren einer UNIX-Maschine heißt `shutdown`. Leider unterscheiden sich die Optionen dieses Befehls geringfügig zwischen BSD und System V.

Bei BSD-Systemen und auch bei Linux hat der Befehl `shutdown` zwei Parameter. Der erste ist eine Zeitangabe, die in jedem Fall angegeben werden muss. Der zweite Parameter ist optional und wird als Nachricht an alle Terminals gesendet. Die Zeitangabe kann einmal als 24-stündige Uhrzeit im Format `hh:mm` angegeben werden. Man kann die Zeit bis zum Shutdown in Minuten angeben. Dieser Zahl ist ein Pluszeichen zur Unterscheidung voranzustellen. Statt `+0` kann auch das Wort `now` verwendet werden.

System V verlangt zwei Optionen bei der Ausführung des Befehls. Hinter der Option `-g` (grace: engl. Gnade) wird die Zeit in Sekunden angegeben, bevor das Herunterfahren beginnt. Die Option `-i` gibt an, welcher Runlevel angesteuert wird. Auch hier kann als Letztes eine Nachricht angegeben werden, die an die Benutzer versandt wird. Wird die Option `-y` mit angegeben, fragt `shutdown` vor dem Herunterfahren nicht noch einmal nach. Siehe zur Unterscheidung von BSD und System V auch: Rainer Schöpf: Shutdown. Auf der Website http://www.uni-mainz.de/~schoepf/sysadm/sysadm_8.html

Gemeinsam ist beiden Systemen, dass sie die Option `-r` für `reboot` und die Option `-h` für `Halt`, also ein Herunterfahren, interpretieren. Letztlich halten sich aber durchaus nicht alle Systeme genau an die oben beschriebene Syntax. So unterscheiden sich die Befehle zum sofortigen Herunterfahren für einen Computer mit Solaris 8 (`sol`) und einen Rechner mit HP-UX 10 (`hpsrv`) durchaus:

```
sol# shutdown -y -g0 -i0
hpsrv# shutdown -h -y 0
```

Wenn Sie die Aufrufparameter Ihrer Maschine kennen lernen wollen, sehen Sie bitte in die Manpage, oder rufen Sie `shutdown` mit der Option `-?` auf. Alternativ verwenden einige Systeme die Befehle `halt` und `reboot`. Überall funktioniert auch `init 0`.

Allen diesen Möglichkeiten ist gemeinsam, dass der Administrator `root` das Herunterfahren veranlassen muss. Das ist durchaus sinnvoll, da ein Produktionsserver nicht einfach von jedem heruntergefahren werden darf. Anders ist die Lage bei Workstations. Da kommt es häufiger mal vor, dass der Anwender auch seine Maschine herunterfahren muss. Hier kann man mit `sudo` (siehe S. `sudo`) arbeiten oder einen Login einrichten, der zum Herunterfahren der Maschine dient (siehe S. `shutdownuser`). Eine sehr elegante Variante findet sich unter Linux. Hier kann in der Datei `/etc/shutdown.allow` hinterlegt werden, welche Benutzer berechtigt sind, `shutdown` aufzurufen. Sie müssen dann allerdings beim Aufruf zusätzlich die Option `-a` angeben.

Es gibt nur wenige Gründe dafür, einen UNIX-Server herunterzufahren. Üblicherweise geschieht dies ausschließlich bei der Installation von Hardware, die man nicht im laufenden Betrieb wechseln kann. Aufgrund einer Konfigurationsänderung ist ein Neustart im Normalfall nicht erforderlich. Es gibt Fälle, in denen ein Neustart schnell und verlässlich testet, ob vorgenommene Änderungen das System auch nach dem nächsten Booten noch korrekt arbeiten lassen.

Unterabschnitte

- [Alles bereit zum Untergang?](#)
- [Wechsel in den Single-User-Modus](#)

« [Konfigurationsdateien](#) | **Administration** | [Alles bereit zum Untergang?](#) »

Alles bereit zum Untergang?

Wenn ein UNIX-Server heruntergefahren wird, ist das keine Privatangelegenheit des Systemadministrators. Das Herunterfahren sollte nur erfolgen, wenn niemand mehr an der Maschine arbeitet. Ein einfacher Befehl `finger` oder `who` gibt Auskunft, ob alle Benutzer von den Terminals abgemeldet sind. Auch Anmeldungen über virtuelle Terminals können so erkannt werden. Virtuelle Terminals sind Netzwerkverbindungen, die den Charakter von Terminalsitzungen haben. Man erkennt sie daran, dass die Benutzer sich mit `/dev/pty`-Devices angemeldet haben.

Schwieriger wird es, wenn auf der Maschine echte Client-Server-Applikationen laufen. Hier helfen Befehle wie `lsof` (siehe S. [lsof](#)), der die offenen Dateien anzeigt, oder `netstat` (siehe S. [netstat](#)), der Netzverbindungen anzeigt.

« [Herunterfahren: shutdown](#) | **[Herunterfahren: shutdown](#)** | [Wechsel in den Single-User-Modus](#) »:

Wechsel in den Single-User-Modus

Der Single-User-Modus hat für die Wartung einer UNIX-Maschine eine besondere Bewandnis. Hier kann man sicher sein, dass kein weiterer Benutzer auf der Maschine arbeitet. Da auch die Netzverbindungen in diesem Zustand nicht aktiv sind, können die Platten auch nicht auf dem Umweg über die Netzdienste von Anwendern verwendet werden. Man kann also in aller Ruhe Festplatten testen oder sonstige Arbeiten durchführen, die die Störung fremder Benutzer nicht vertragen. Das Herunterfahren in den Single-User-Modus ist aus Sicht der Benutzer allerdings nichts anderes als ein normales Herunterfahren. Entsprechend muss man auch hier schauen, dass man niemandem seine Prozesse unterbricht. Der Wechsel selbst erfolgt einfach durch Aufruf von:

```
init 1
```

Zurück in den bisherigen Modus kommt man durch Schließen der Shell mit ctrl-D oder durch `init 2` bzw. `init 3`.

[« Alles bereit zum Untergang? | Herunterfahren: shutdown | Benutzerverwaltung »](#)

Benutzerverwaltung

Die Benutzerverwaltung gehört zu den Routinetätigkeiten des Administrators. Für jeden Benutzer wird eine Kennung, ein Passwort und ein Bereich auf der Platte angelegt, in dem der er arbeiten kann. Es treten Aspekte der Einbruchssicherheit auf, und hin und wieder müssen den Benutzern auch Grenzen gesetzt werden.

Unterabschnitte

- [Passwortverwaltung unter UNIX](#)
 - [Die Benutzerdatei /etc/passwd](#)
 - [Verborgene Passwörter: shadow](#)
 - [Benutzerpflege automatisieren](#)
 - [Grundeinstellungen: /etc/profile](#)
 - [ulimit](#)
 - [umask](#)
 - [Umgebungsvariablen](#)
 - [Verzeichnisprototyp: /etc/skel](#)
 - [Gruppenverwaltung](#)
 - [Benutzerüberwachung](#)
 - [Accounting](#)
 - [who und finger](#)
 - [Kurzfristiger Benutzerwechsel: su](#)
 - [Administrationsaufgaben starten: sudo](#)
 - [Pseudob Benutzer zum Shutdown](#)
-

Passwortverwaltung unter UNIX

Informationen über die Anwender werden in der Datei `/etc/passwd` gehalten. Hier steht für jeden Benutzer, wo das Heimatverzeichnis liegt, welche Shell gestartet wird, und hier wird die User-ID und die Group-ID gespeichert. Traditionsgemäß enthält die Datei `/etc/passwd` auch das verschüsselte Passwort.

Die Benutzerdatei /etc/passwd

In den älteren UNIX-Systemen manipulierte man für einen Benutzereintrag direkt die Datei /etc/passwd. Dieser Weg steht prinzipiell auch heute noch offen, sofern die Passwörter nicht zentral im Netz unter NIS (siehe S. nis) verwaltet werden.

Die /etc/passwd zeigt, wie der Benutzer eines UNIX-Systems definiert ist. Ein Eintrag in der /etc/passwd hat folgenden Aufbau:

Name:Passwort>User-ID:Group-ID:Kommentar:Verzeichnis:Shell

Hier folgt ein Beispiel für eine /etc/passwd:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
daemon:x:2:2:daemon:/sbin:/bin/bash
lp:x:4:7:lp daemon:/var/spool/lpd:/bin/bash
ftp:x:40:2:ftp account:/usr/local/ftp:/bin/bash
named:x:44:44:Nameserver Daemon:/var/named:/bin/bash
nobody:x:65534:65534:nobody:/var/lib/nobody:/bin/bash
arnold:x:501:100:Arnold willemer:/home/arnold:/bin/bash
andrea:x:502:100::/home/andrea:/bin/bash
```

Die Einträge bedeuten im Einzelnen:

- [Name] Der Benutzername wird beispielsweise zur Anmeldung am System verwendet. Unter diesem Namen wird der Benutzer bei Rechtezuordnungen angesprochen. Häufig wird dazu der Nachname, der Vorname oder eine Mischung aus beidem verwandt. Der Benutzername ist alles andere als geheim und sollte leicht der wirklichen Person zuzuordnen sein. Bei einigen Systemen ist er auf acht Buchstaben begrenzt.
- [Passwort] In Systemen ohne Shadow-Datei steht hier das verschlüsselte Passwort. Es handelt sich um einen Falltür-Algorithmus. Das bedeutet, dass man aus dem Passwort im Klartext zwar leicht die Verschlüsselung ermitteln kann, dass der umgekehrte Weg aber nicht möglich ist. An sich ist das Verfahren recht sicher. Mit Hilfe einer Lexikondatei und einem Namensregister kann man aber schlechte Passwörter leicht knacken.

Beim Anlegen eines neuen Benutzers lässt man diesen Eintrag frei. Ein leerer Eintrag erlaubt den Zugang ohne Passwort. Gleich anschließend sollte der neue Benutzer den Eintrag durch den Aufruf des Befehls passwd füllen.

Soll der Benutzer für interne Zwecke angelegt werden, dann ist es durchaus möglich, dass man vermeiden will, dass sich jemand unter dieser Kennung anmeldet. Das erreicht man, indem man einen Stern oder ein X als Passwort einsetzt. Der Verschlüsselungsalgorithmus kann nämlich niemals einen einstelligen Eintrag an dieser Stelle erzeugen. Damit gibt es kein Passwort, das hierzu passt.

Ein kleines x an dieser Stelle in allen Einträgen deutet darauf hin, dass das System eine Shadow-Datei für die Passwörter verwendet.

- [User-ID] Jeder Benutzer hat seine eigene Nummer. Normale Benutzer werden auf einigen Systemen ab 50, ab 100 oder neuerdings ab 500 angelegt. Die kleineren Nummern sind teilweise für Systemdienste festgelegt, so ist das Administrationskennwort root mit der User-ID 0 verbunden. Der Eigentümer von Dateien wird in den i-nodes mit der User-ID gekennzeichnet.
- [Group-ID] Jeder Benutzer gehört zu mindestens einer Gruppe. Die Hauptgruppe, zu der der Benutzer gehört, wird hier eingetragen. Der Benutzer kann in der Datei /etc/group auch in weiteren Gruppen angemeldet werden.
- [Kommentar] Hier wird im Klartext eingetragen, wer der Benutzer ist. Der Eintrag hat informativen Charakter.
- [Verzeichnis] Das Heimatverzeichnis des Benutzers. Hier hat der Anwender seinen Arbeitsbereich, und hier wird er auch landen, sobald er sich eingeloggt hat. Auch Einstellungen wie die .profile-Datei stehen hier. Der neue Benutzer muss in das Verzeichnis wechseln und es lesen können (x+r). In den meisten Fällen werden die Benutzer es schätzen, wenn sie in ihrem Bereich auch schreiben können.
- [Shell] Hier wird normalerweise die Shell eingetragen, die für den Benutzer beim Einloggen gestartet wird. Je nach Geschmack kann neben der klassischen Bourne-Shell (sh) auch die Kornshell (ksh), die C-Shell (csh) oder die Bourne-Again-Shell (bash) eingesetzt werden. Lediglich für den root sollte man eine Shell wählen, die auch dann zur Verfügung steht, wenn nur die Bootpartition ansprechbar ist. Die Shell ist mit vollem Pfadnamen einzutragen.

Um neue Benutzer direkt in die /etc/passwd-Datei einzutragen, kopiert man am einfachsten einen bisherigen Eintrag, korrigiert die benutzerbezogenen Daten, insbesondere den Benutzernamen und erstellt ein neues Heimatverzeichnis. Zum Beispiel:

```
meier::237:106::/home/meier:/bin/sh
```

Für den Benutzer wird ein Verzeichnis eingerichtet. In unserem Beispiel:

```
mkdir /home/meier
```

Gegebenenfalls wird der Benutzer in die Gruppendatei /etc/group aufgenommen (siehe S. group). Zum Beispiel:

```
projekt_a::106:petersen,meier
post::107:schulz,mueller,meier
```

Das Heimatverzeichnis des neuen Benutzers wird mit dessen Zugriffsrechten versehen. Zum Beispiel:

```
gaston# chown meier /home/meier
gaston# chgrp projekt_a /home/meier
```

Mit dem Befehl passwd wird schließlich das Passwort gesetzt:

```
passwd meier
```

Wenn allerdings Schattenpasswörter in der Datei /etc/shadow verwendet werden, muss vor dem ersten Aufruf von passwd noch ein Eintrag in dieser Datei vorgenommen werden. Ansonsten landet das verschlüsselte Passwort doch wieder in /etc/passwd.

Verborgene Passwörter: shadow

Die Datei `/etc/passwd` enthält heute normalerweise keine verschlüsselten Passwörter mehr. Diese werden im System in einer zweiten Datei namens `/etc/shadow` abgelegt, die nur noch von root lesbar ist. Der Grund für diese Maßnahme ist, dass viele Anwender Passwörter verwenden, die leicht zu knacken sind. Das ist vor allem der Fall, wenn das Passwort Wörtern entspricht, die man in einem Lexikon findet oder die übliche Vornamen sind.

Ein gängiger Angriff auf eine Maschine erfolgte dadurch, dass man die für jeden lesbare Datei `/etc/passwd` kopierte. Anschließend ließ man ein kleines Programm über ein Wörterbuch laufen, das jedes Wort verschlüsselt und die Verschlüsselung mit der Zeichenfolge in der `/etc/passwd` vergleicht. Ein solches Programm zu schreiben ist nicht weiter schwierig, da das Verfahren, mit dem UNIX seine Passwörter verschlüsselt, öffentlich bekannt ist.

So wie man einen Benutzer in der `/etc/passwd` von Hand eintragen kann, ist das auch in der `/etc/shadow` möglich. Am einfachsten ist es auch hier, einen existierenden Eintrag zu kopieren und an den neuen Benutzer anzupassen. Wie in der `passwd`-Datei werden auch hier die Einträge durch Doppelpunkte voneinander getrennt. Dabei stehen folgende Einträge hintereinander:

- Die Benutzerkennung, die auch in der `/etc/passwd`-Datei in der ersten Spalte steht
- Das Passwort
- Der Tag der letzten Änderung des Passworts. Das Datum wird als Zahl der seit dem 1.1.1970 vergangenen Tagel im Gegensatz zur sonst bei UNIX üblichen Codierung eines Zeitpunkts als die Anzahl der Sekunden seit dem 1.1.1970 wird hier tatsächlich die Anzahl der Tage verwendet. codiert
- Die Anzahl der Tage, nach denen das Passwort erstmals geändert werden darf
- Die Anzahl der Tage, nach denen das Passwort geändert werden muss
- Die Anzahl der Tage, die der Benutzer vor dem Ablauf des Passworts gewarnt wird
- Eine Frist in Tagen, die angibt, wann der Ablauf des Passworts geschlossen wird
- Das Datum, an dem der Zugang geschlossen wurde in Tagen seit dem 1.1.1970
- Reserviertes Feld

Beispiel eines Eintrags:

```
arno1d::11545:0:99999:7:0::
```

Man sieht, dass man durch die `/etc/shadow`-Datei neben der Sicherung der verschlüsselten Passwörter auch die Möglichkeit gewinnt, Passwörter ablaufen zu lassen. Wenn Sie diese Option einsetzen möchten, denken Sie daran, dass ein häufiger Wechsel kontraproduktiv sein kann. Sobald die Anwender den Überblick verlieren, werden sie die Passwörter aufschreiben, anstatt sie sich zu merken, oder sie lassen sich Kreationen wie `januar1`, `februar2` und so fort einfallen.

Benutzerpflege automatisieren

Bei den meisten Systemen existiert ein Dienstprogramm namens `useradd`. Manchmal heißt es auch `adduser`. Damit kann man einen Benutzer anlegen. Das Programm prüft die Verträglichkeit des Eintrags und setzt sinnvolle Vorgaben, damit man nicht so viel tippen muss. Im folgenden Beispiel wird ein Benutzer mit der Kennung `testperson` angelegt.

```
gaston# useradd testperson
gaston# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
...
nobody:x:65534:65534:nobody:/var/lib/nobody:/bin/bash
willemer:x:500:100:Arnold willemer:/home/willemer:/bin/bash
arnold:x:501:100:Arnold willemer:/home/arnold:/bin/bash
andrea:x:502:100:./home/andrea:/bin/bash
testperson:x:503:100:./home/testperson:/bin/bash
gaston# ls /home
.  .. andrea arnold cvsroot t345-1.33 willemer
gaston# mkdir /home/testperson
gaston# chown testperson /home/testperson
```

Durch `useradd` wird der Eintrag in der `/etc/passwd` generiert. Das zeigt die Ausgabe der Datei. Das Kommentarfeld und das Passwort lässt das Programm frei. Das Heimatverzeichnis des Benutzers wird zwar in der Datei `/etc/passwd` benannt, muss aber erst noch angelegt werden. Solche Tools sind durchaus praktisch, man sollte aber genau wissen, wo sie ihre Grenzen haben und wann man von Hand nacharbeiten muss. Analog gibt es auch kleine Programme, die eine Änderung oder das Austragen des Benutzers durchführen.

Alle Administrationstools erlauben, dass man auch Benutzer verwalten kann. Im Allgemeinen leisten sie mehr als die oben genannten Skripten.

Wenn Sie plötzlich Probleme haben, im Verzeichnis `/home` einen Benutzer anzulegen und dies auch nicht durch das Administrationstool möglich ist, kann es sein, dass ein Automount-Dämon (siehe S. automount) das Verzeichnis `/home` verwaltet. In diesem Fall ist zu prüfen, ob die Heimatverzeichnisse im Netz per Automount verwaltet werden. Falls das nicht beabsichtigt ist, muss man diesen Dämon deinstallieren, bevor man im Verzeichnis `/home` Benutzerverzeichnisse anlegt.

Unterabschnitte

- [ulimit](#)
 - [umask](#)
 - [Umgebungsvariablen](#)
-

Grundeinstellungen: /etc/profile

Die Datei /etc/profile wird beim Einloggen jedes Benutzers gestartet, der nicht die C-Shell verwendet. Für die C-Shell heißt die entsprechende Datei /etc/csh.chsrc. Die Unterscheidung ist insofern sinnvoll, als die C-Shell eine andere Syntax für das Setzen von Variablen hat. In diesen beiden Dateien können Einstellungen vorgenommen werden, die für alle Benutzer gelten sollen.

ulimit

Mit dem Befehl `ulimit` können den Anwendern Beschränkungen aufgelegt werden, die die Größen von einigen Ressourcen betreffen. Die Optionen von `ulimit` bewirken:

[Optionen von `ulimit`]

Option	Wirkung
-a	Anzeige aller aktuellen Limits
-c	maximale Größe eines Core-Dumps
-d	maximale Größe des Datasegments eines Prozesses
-f	maximale Größe einer Datei
-l	maximaler gesperrter Hauptspeicherbereich
-n	maximale Anzahl offener Dateien (nicht alle Systeme)
-s	maximale Größe des Stacks eines Prozesses
-t	maximale CPU-Zeit in Sekunden
-v	maximale Größe des virtuellen Speichers

Die Größe einer Datei zu beschränken, um den Plattenplatz zu limitieren, dürfte nicht der richtige Weg sein, da viele kleine Dateien genauso viel Platz einnehmen können wie eine große. Ein solches Problem löst man besser mit Quotas (siehe S. `quota`).

Die hier festgelegten Einschränkungen sollten allgemein bekannt gemacht werden. Im Allgemeinen wird das Überschreiten der Limits zu einem Programmabbruch führen, und die Ursache kann gegebenenfalls schwer zu ermitteln sein.

umask

Der Befehl `umask` setzt eine Maske auf diejenigen Rechte, die bei der Erzeugung einer Datei nicht gesetzt werden. Ein typischer Wert ist 022. Das bedeutet, dass die Gruppe und die Welt keine Schreibrechte auf die Dateien und Verzeichnisse haben. Letztlich kann der Anwender natürlich die Rechte nach ihrer Erzeugung mit einem `chmod` beliebig setzen. Allerdings weiß er dann auch, was er tut. Der Befehl `umask` dient also in erster Linie dazu, den Anfänger davor zu beschützen, dass er

versehentlich Dateien anlegt, die dann von aller Welt verändert werden können.

Umgebungsvariablen

Die Vorbelegung von Umgebungsvariablen, die für alle Benutzer der Maschine gelten soll, kann ebenfalls in `/etc/profile` erfolgen. Eine der wichtigsten ist die Variable `PATH`, die beschreibt, wo die startbaren Programme zu finden sind. Andere Einstellungen, wie der Standardprinter oder Einstellungen zur grafischen Umgebung können hier systemweit vorgegeben werden. Aber auch Variablen, die als Schalter für Anwendungssoftware dienen, können hier definiert werden.

Es gibt aber einen guten Grund dafür, Umgebungsvariablen in den lokalen rc-Dateien zu belegen. Auf diese Weise sieht der Anwender die Vorbelegung der Variablen, und er kann sie leicht verändern.

« [Benutzerpflege automatisieren](#) | **[Benutzerverwaltung](#)** | [Verzeichnisprototyp: /etc/skel](#) »

Verzeichnisprototyp: /etc/skel

Das Heimatverzeichnis eines neuen Benutzers ist typischerweise nicht völlig leer, sondern enthält diverse rc-Dateien und Konfigurationsdateien. Viele dieser Dateien beginnen mit einem Punkt, damit der Benutzer sie nicht versehentlich durch `rm *` löschen kann. Sie sind auch für `ls` unsichtbar, sofern nicht die Option `-a` verwendet wird.

Damit sie nicht für jeden Benutzer mühselig neu erstellt werden müssen, gibt es unter `/etc` oder `/usr/share` ein Verzeichnis namens `skel`, dessen Inhalt man einfach in das neu angelegte Verzeichnis kopieren kann. So würde das Anlegen eines neuen Benutzerverzeichnisses durch folgende Befehle erfolgen:

```
gaston# mkdir /home/meier
gaston# chown meier /home/meier
gaston# chgrp projekt_a /home/meier
gaston# cp /etc/skel/.??* /home/meier
gaston# cd /home/meier
gaston# chmod 644 .??*
gaston# chown meier .??*
gaston# chgrp projekt_a .??*
```

Der Grund für die Maske `.??*` ist, dass verhindert werden muss, dass das Verzeichnis `..` ebenfalls dem Benutzer `meier` zugeordnet wird. Die beiden Fragezeichen gewährleisten, dass nach dem Punkt noch mindestens zwei Zeichen folgen. Das trifft für die Dateien in `/etc/skel` normalerweise zu. Alternativ könnte als Maske auch `[a-zA-Z]*` verwendet werden. Diese gewährleistet, dass nach dem Punkt ein Buchstabe folgen muss.

Gruppenverwaltung

Eine Gruppe ist eine Menge von Benutzern. Durch die Gruppe ergibt sich die Möglichkeit, die Berechtigungen von Dateien so zu wählen, dass ein Team gemeinsam auf Ressourcen zugreifen kann, ohne dass die Daten gleich allen Benutzern öffentlich angeboten werden.

Die Verfahrensweise ist simpel. Die Gruppen werden in der Datei `group` im Verzeichnis `/etc` verwaltet. Hinter dem Gruppennamen stehen die Benutzerkennungen der Mitglieder der Gruppe. Eine Datei, die nur für die Gruppe gedacht ist, wird durch den Befehl `chgrp` zum Eigentum der Gruppe. Mit dem Befehl `chmod` werden die Zugriffsrechte der Gruppe gesetzt.

Jeder Benutzer gehört zu einer Standardgruppe, die in der `passwd`-Datei festgelegt wird. Das bedeutet, dass Dateien, die er erzeugt, automatisch dieser Gruppe zugeordnet werden. Der Benutzer kann aber in beliebig vielen anderen Gruppen eingetragen sein. Dazu wird in der Datei `/etc/group` sein Name an die Liste der Mitglieder der jeweiligen Gruppe gehängt:

```
dialout:x:16:root,arnold,andraea  
prog:x:101:arnold,ralf
```

Zur Gruppe `dialout` gehören `root`, `arnold` und `andraea`, zur Gruppe `prog` `arnold` und `ralf`. An der dritten Stelle steht die eindeutige Nummer der Gruppe. Die Definition eigener Gruppen sollte man bei 101 beginnen.

Bis hierher reichen die Gemeinsamkeiten der Systeme, die immerhin durchaus eine klar geregelte Gruppenbildung ermöglicht.

Mit dem Befehl `newgrp` kann man die Gruppe wechseln. Seine Syntax ähnelt dem Kommando `su`. Sofern der Benutzer in der Zielgruppe eingetragen ist, wird die reale Gruppe gewechselt. Gehört er nicht zur Gruppe und besitzt die Gruppe ein Passwort, wird er danach gefragt. Ist er nicht in der Gruppe per `/etc/group` eingetragen und hat die Gruppe kein Passwort, ist ein Wechsel nicht möglich. Leider reagieren nicht alle Systeme gleich. In einigen Systemen wird trotz des Eintrags in einer anderen Gruppe nach dem Passwort gefragt, in anderen Gruppen ist ein Passwort gar nicht vorgesehen.

Obwohl in der zweiten Spalte der `group`-Datei Platz für ein Passwort vorhanden ist, verfügen einige System über keinen Befehl, um dieses Passwort zu besetzen. Unter Linux wird beispielsweise der Befehl `gpasswd` oder der normale Befehl `passwd` mit der Option `-g` verwendet. Der Befehl `gpasswd` ermöglicht es `root` mit der Option `-A`, einen Gruppenadministrator für jede Gruppe festzulegen, der dann ebenfalls mit dem Befehl `gpasswd` Mitglieder hinzufügen oder austragen kann.

Unterabschnitte

- [Accounting](#)
 - [who und finger](#)
-

Benutzerüberwachung

Das Überwachen von Benutzern hat einen schlechten Beigeschmack. Normalerweise befasst sich ein Administrator nicht sehr gern damit. Allerdings ist es wichtig zu wissen, welche Möglichkeiten existieren, wenn ein Benutzer das in ihn gesetzte Vertrauen missbraucht.

Accounting

Das Accounting ist das Berechnen der Kosten, die ein Benutzer auf dem System verursacht. In Zeiten, da eine UNIX-Maschine ein Vermögen kostete, lohnte es sich durchaus, den hoch bezahlten Administrator berechnen zu lassen, welchen Kostenanteil welcher Benutzer verursachte. Heute ist der Kostenaspekt so weit zurückgegangen, dass ein Accounting fast nicht mehr durchgeführt wird. In einer Hinsicht ist es aber immer noch interessant: Da dabei alle Aktivitäten der Benutzer protokolliert werden, hat man Informationen zur Hand, wenn auf der Maschine Unregelmäßigkeiten vorkommen.

Das Programm `accton` kontrolliert das Accounting. Wird ihm eine Datei als Argument mitgegeben, startet das Accounting und protokolliert in diese Datei. Als Datei wird üblicherweise `acct` oder `pacct` im Verzeichnis `/var/adm/` verwendet. Die Datei muss beim Aufruf existieren. In der Datei werden die Anzahl der Programmaufrufe, die verbrauchte CPU-Zeit, die I/O-Operationen und die Speicherbenutzung für jeden Anwender protokolliert. Das Accounting wird durch den Aufruf von `accton` ohne Parameter abgeschaltet.

Um die Verbindungszeiten zu protokollieren, gibt es die Datei `wtmp` im Verzeichnis `/var/adm`. Sie wird von `init` und `login` gefüllt und kann recht schnell recht groß werden. Darum ist es sinnvoll, von Zeit zu Zeit zu kontrollieren, ob die Datei nicht gestutzt werden sollte (siehe S. `longprotocol`). Braucht man die Verbindungsdaten nicht, kann man sie einfach löschen. Existiert die Datei nicht, wird sie auch nicht von `init` oder `login` gefüllt. Zum Auswerten der Datei zum Zwecke des Accounting dient das Programm `ac`. Mit der Option `-u` wertet das Programm nach Benutzer, mit der Option `-d` nach dem Datum aus.

who und finger

`who` zeigt an, welche Benutzer an welchen Terminals angemeldet sind. `finger` zeigt alle angemeldeten Benutzer. Für jeden wird angezeigt, seit wann er angemeldet ist und seit wann keine Aktivität mehr feststellbar ist. Auf manchen Systemen gibt es den Befehl `whodo`. Damit kann man sehen, welches Programm welcher Benutzer aktuell gestartet hat.

finger und who werten die Datei utmp aus, die sich normalerweise in /var/run oder in /var/log befindet. Hier sollen sich alle für das Einloggen zuständigen Programme eintragen. Allerdings gibt es da Unterschiede in der Interpretation. So nimmt xterm einen Eintrag vor, konsole, die xterm-Variante von KDE, nimmt dagegen keinen Eintrag vor und begründet dies damit, dass es sich bei einer xterm-Sitzung ja nicht um einen weiteren Benutzer des Systems handelt.

« [Gruppenverwaltung](#) | [Benutzerverwaltung](#) | [Kurzfristiger Benutzerwechsel: su](#) »:

Kurzfristiger Benutzerwechsel: su

Obwohl der Befehl `su` (set user) oft als Abkürzung für »superuser« bezeichnet wird, dient er dazu, während der aktuellen Terminalsitzung die eigenen Identität in eine beliebige andere zu wechseln. Wenn Sie `su` ohne Parameter aufrufen, werden Sie nach dem Administrationspasswort gefragt und mutieren zum Benutzer `root`. Der Wechsel zu `root` wird normalerweise protokolliert.

Beim einfachen Aufruf von `su` werden die profile-Skripten nicht durchlaufen und das Arbeitsverzeichnis wird nicht gewechselt. Als Ergebnis hat man zwar `root`-Rechte, aber die Umgebung für die Werkzeuge des `root` sind nicht gesetzt. Insbesondere stimmt die Variable `PATH` nicht, sodass die Programme im Verzeichnis `sbin` beispielsweise nicht gefunden werden. Will man sich vollständig als `root` anmelden, verwendet man `su -`. Dann wird ein regulärer Login vollzogen.

Man kann bei `su` auch einen anderen Benutzernamen als Parameter angeben und wechselt dann seine Identität. Normale Benutzer müssen dann natürlich das Passwort des Zielbenutzers angeben. Nur `root` muss das nicht. Auf diese Weise ist es möglich, Benutzer anzulegen, die kein Passwort besitzen. Diese können sich zwar nicht am Terminal anmelden, aber `root` kann mit `su` deren Identität annehmen, da er kein Passwort braucht. Ein Beispiel ist der Benutzer `news`, der für die Administration des Newsgroupservers (siehe S. `news`) gebraucht wird. Dieser Benutzer kann einen Stern als Passwort bekommen. Auf diese Weise kann sich niemand über diesen Account von außen anmelden. Die Newsgroup wird normalerweise sowieso vom Systemadministrator gepflegt. Er kann sich leicht per `su news` anmelden, ohne dass ein Passwort existiert.

[Varianten des Aufrufs von `su`]
L|L Befehl & Wirkung

`su` & `root`-Login ohne Umgebungswechsel

`su -` & `root`-Login inklusive Umgebungswechsel

`su - username` & Login als anderer Benutzer

Administrationsaufgaben starten: sudo

Es gibt diverse Aufgaben, für die man die Berechtigung von root braucht, die aber durchaus auch in die Hände einzelner Benutzer gelegt werden können. Dennoch möchte man diesen Benutzern deswegen nicht gleich das Passwort von root verraten. Diese Lücke schließt das Programm sudo.

Der Aufrufer stellt den Befehl sudo einem Kommando voran. Das bewirkt, dass das Kommando unter einer fremden Benutzerkennung läuft, typischerweise der des Administrators. Die einzelnen Befehle und die Anwender, die sie benutzen dürfen, werden vorher festgelegt. Jede Benutzung von sudo wird exakt protokolliert.

Die Konfigurationsdatei /etc/sudoers hält die Information, welche Benutzer den Befehl sudo verwenden dürfen. Sie wird nicht direkt, sondern mit dem Programm vi sudo als root editiert. Die Datei sudoers wird in mehrere Gruppen eingeteilt. Die ersten Gruppen haben nur den Zweck, Makros zu bilden, um die Rechte einfacher zu beschreiben. Als Benutzerprivilegien ist von vornherein eingetragen, dass root auf allen Rechnern alles darf. Neu hinzu kommt, dass der Benutzer arnold seine Datensicherung auf gaston ausführen darf.

```
# Host alias specification
```

```
# User alias specification
```

```
# Cmnd alias specification
```

```
# User privilege specification
```

```
root    ALL=(ALL) ALL
```

```
arnold  gaston = /root/msg
```

```
arnold  gaston = /home/arnold/bin/cddasi
```

Da das Skript cddasi per cdrecord (siehe S. cdrecord) auf den CD-Brenner zugreift, braucht man die root-Rechte. Man kann das Problem lösen, indem man der Gerätedatei des Brenners freie Rechte mit chmod gibt. Damit ist das Gerät für jeden zu jeder Verwendung freigegeben. Man kann auch den in den Manpages von cdrecord vorgeschlagenen Weg gehen und für den Befehl cdrecord das SUID-Bit (siehe S. suid) setzen:

```
chmod 4755 cdrecord
```

Damit würde immerhin erreicht, dass der Zugriff auf den Brenner nur über die Anwendung cdrecord kontrolliert erfolgen kann. Die Freigabe per sudo ermöglicht aber sogar die Einschränkung auf das Skript cddasi. Der Anwender darf mit dem Brenner also ausschließlich seine Datensicherung durchführen. Er kann beispielsweise keine Audio-CDs brennen. Des Weiteren ermöglicht sudo, dieses Recht einzelnen aufgeführten Anwendern zu gewähren. Und zu guter Letzt wird die Verwendung des Befehls auch noch protokolliert.

Will der Benutzer arnold das Skript cddasi ausführen, gibt er das Kommando mit vorangestellten sudo ein:

```
sudo /bin/cddasi
```

sudo fragt nun nach einem Passwort. Dies ist aber nicht das Passwort von root, sondern das des angemeldeten Benutzers. Damit wird vermieden, dass jemand die nicht geschlossene Konsole bei kurzfristiger Abwesenheit für privilegierte Aufgaben nutzen kann.

In der Protokolldatei des syslog-Dämons (meist /var/log/messages, siehe S. syslog) wird protokolliert, dass der Benutzer arnold das Skript als root ausgeführt hat. Hier sind zwei Protokolleinträge dargestellt. Der erste wurde ohne Berechtigung durchgeführt. Im zweiten wurde der Eintrag in der Datei sudoers eingetragen: Aus drucktechnischen Gründen wurden die Zeilen umbrochen.:

```
Mar  9 05:45:30 gaston sudo:  arnold : command not allowed ;
                TTY=pts/5 ; PWD=/home/arnold ; USER=root ;
                COMMAND=/home/arnold/bin/cddasi
Mar  9 05:46:18 gaston sudo:  arnold : TTY=pts/5 ;
                PWD=/home/arnold ; USER=root ; COMMAND=bin/cddasi
```

Wie oben schon erwähnt wurde, kann man in der Datei sudoers auch mit Makros arbeiten. Dabei können mehrere Benutzer, Maschinen und Kommandomakros gebildet werden und in der Rechtevergabe verwendet werden:

```
User_Alias      DEPUTIES = wim, wum, wendelin
Host_Alias      SERVERS = gaston, asterix, silver
Cmnd_Alias      HALT = /usr/sbin/halt, /usr/sbin/fasthalt
DEPUTIES SERVERS = HALT
```

Hier können alle Benutzer, die unter DEPUTIES aufgeführt sind, alle Server herunterfahren. Vielleicht ist es irritierend, dass in der Datei auch die root-Rechte fremder Rechner festgelegt werden. Es ist natürlich nicht möglich, auf einem Rechner festzulegen, welche Rechte ein Anwender auf einem andern Rechner bekommt. Jeder Rechner kann seine root-Rechte nur selbst vergeben. Der Grund, dass hier auch Hostnamen genannt werden, besteht darin, dass die Datei sudoers per NIS (siehe S. nis) verteilt werden kann und dann jeder Rechner anhand des Rechnernamens nur den Teil übernimmt, der für ihn gedacht ist.

Durch den Aufruf von `sudo -l` kann ein Benutzer sehen, welche Kommandos er unter sudo ausführen kann.

Das Programm sudo hat noch vielfältige Optionen. So kann für einzelne Kommandos oder Gruppen die Passworteingabe abgeschaltet werden. Es kann konfiguriert werden, dass das Protokoll per Mail versandt wird. Diese Informationen findet man in den umfangreichen Manpages von sudo und sudoers.

« [Kurzfristiger Benutzerwechsel: su](#) | [Benutzerverwaltung](#) | [Pseudob Benutzer zum Shutdown](#) »

Pseudob Benutzer zum Shutdown

Bei einer Workstation kann es wichtig sein, dass auch nicht privilegierte Anwender die Maschine herunterfahren können. Der Vorgang sollte dann so einfach wie möglich sein, denn je umständlicher das Herunterfahren ist, desto eher besteht die Neigung, die Maschine einfach abzuschalten.

Eine elegante Lösung ist es, einen Benutzer anzulegen, dessen einziger Zweck das Herunterfahren ist. Der Benutzer kann beispielsweise shutdown heißen, und damit es sich jeder merken kann, bekommt er auch das Passwort shutdown. Nachdem er angelegt wurde, wird in die Datei .profile im Heimatverzeichnis ganz zu Anfang eine einzige Zeile geschrieben:

```
sudo shutdown -h 0
```

Damit wird direkt nach dem Anmelden der Befehl shutdown aufgerufen. Da dies nur dem Administrator root erlaubt ist, wird sudo verwendet. Nun muss der Benutzer shutdown noch einen Eintrag in die Datei /etc/sudoers bekommen, der ihm erlaubt, den besagten shutdown auszuführen:

```
shutdown    localhost=NOPASSWD:/sbin/shutdown -h 0
```

Das Attribut NOPASSWD ist wichtig, da die Ausführung von .profile nicht interaktiv ist. Damit wird für das folgende Kommando kein Passwort abgefragt.

Alternativ kann man für Benutzer shutdown in der Datei /etc/passwd als Shell das Programm shutdown eintragen. Dazu muss der komplette Pfad von shutdown angegeben werden, der durch das Kommando which leicht ermittelt werden kann. Beim Anmelden als Benutzer shutdown wird die Shell gestartet, die in diesem Fall das Kommando zum Herunterfahren des Rechners ist. Einen kleinen Haken gibt es noch: Da shutdown immer noch ein privilegiertes Kommando ist, muss auch der Benutzer shutdown ein privilegierter Benutzer sein. Man erreicht dies am einfachsten, indem man als User-ID wie bei root eine 0 einträgt.

```
shutdown:c1ShgL2tYojh.:0:0::/tmp:/sbin/shutdown
```

Hardwarezugriff unter UNIX: /dev

UNIX wurde von vornherein als ein portables Betriebssystem angelegt. Das bedeutet, dass man UNIX auf die unterschiedlichsten Hardwareumgebungen übertragen kann. Dazu ist es notwendig, die Hardwareabhängigkeiten auf einzelne Module zu beschränken, die zum Betriebssystem hin eine immer gleiche Schnittstelle bieten, also eine Abstraktion der speziellen Hardware liefern. Programme, die zwischen der konkreten Hardware und dem Betriebssystem vermitteln, nennt man Treiber.

Unterabschnitte

- [Aufgaben eines Treibers](#)
 - [Gerätedateien](#)
 - [Umgang mit Gerätedateien](#)
 - [Gerätenamen](#)
-

Aufgaben eines Treibers

Hardware wird in den meisten Fällen durch Controllerbausteine gesteuert. Diese Controller besitzen Register, die von dem Treiber mit Werten gefüllt werden, um Aktionen auszulösen, oder die ausgelesen werden, um den Zustand der dahinter befindlichen Hardware zu ermitteln. Die Registerzugriffe steuern also die Hardware und sind vom speziellen Controllerbaustein abhängig. Die Aufgabe, ein Paket ins Netzwerk zu senden, bedeutet, dass die Adresse, an der der zu sendende Speicherinhalt liegt, in ein Register gestellt wird. In ein weiteres Register wird dann die Nummer geschrieben, die den Sendebefehl auslöst. Unterschiedliche Hardware benötigt unterschiedliche Parameter. Beim Schreiben des Speicherinhaltes auf die Platte muss der Hardware zusätzlich mitgeteilt werden, an welche Stelle der Platte der Speicherinhalt geschrieben werden soll. Natürlich unterscheidet sich auch das Protokoll des Controllers für Netzwerkadapter von dem eines Festplattencontrollers. Diese Details verbirgt der Treiber vor dem Betriebssystem.

Die Kommandoübermittlung an den Controller stellt den einen Teil der Aufgabe eines Treibers dar. Die andere Aufgabe ist die Verarbeitung von Interrupts. Soll beispielsweise ein Block von einer Platte gelesen werden, vergeht eine gewisse Zeit, bis der Inhalt im Speicher angekommen ist. Diese Zeit wird UNIX dazu verwenden, andere Prozesse abzuarbeiten. Da der Prozess, der den Speicherblock von der Platte lesen will, erst weiterarbeiten kann, wenn die Leseanforderung bearbeitet wurde, wird er in eine Warteschlange gestellt. Zu irgendeinem Zeitpunkt ist dann der Controller mit dem Lesen des Plattenblocks fertig und die Daten können abgeholt werden. Daraufhin sendet der Controller einen Interrupt (engl. Unterbrechung) an das Betriebssystem. Der Treiber hat beim ersten Start seine Interruptbehandlungsroutine zur Bearbeitung des Interrupts eingetragen, sodass jetzt beim Eintreffen des Interrupts die entsprechende Treiberfunktion aktiviert wird. Diese liest die Registerinhalte des Controllers und stellt fest, ob der Zugriff Erfolg hatte. Anschließend befreit die Interruptfunktion des Treibers den Prozess aus der Warteschlange, übergibt ihm Speicher und ggf. ein Fehlernummer und beendet sich wieder. Der anfordernde Prozess ist nun wieder lauffähig und kann die Ergebnisse weiterverarbeiten.

Gerätedateien

Die für den Benutzer sichtbare Schnittstelle des Treibers ist die Gerätedatei, die im Verzeichnis /dev liegt. Dabei handelt es sich keineswegs um eine gewöhnliche Datei. Man kann Gerätedateien sowohl lesen als auch schreiben. Die Lese- und Schreibkommandos gehen dann als solche an den zuständigen Treiber. Er wird die Daten an »sein« Gerät weiterleiten und die entsprechende Registeransteuerung durchführen. Einem Anwender werden diese Schnittstellen normalerweise nicht zugänglich gemacht, weil beispielsweise der Plattentreiber nur Blöcke von und zur Festplatte transportieren kann, aber keine Ahnung von Dateien oder Verzeichnissen hat. Der naive Zugriff auf diese Schnittstelle könnte also die Struktur der Festplatte völlig durcheinander bringen.

Neben dem Schreiben und Lesen ist es auch nötig, das Gerät in spezifischer Art zu steuern. Dies wird über den Systemaufruf `ioctl()` erreicht. Damit können Programme unter UNIX den Treiber direkt ansprechen. Um mit dem Treiber zu korrespondieren, muss der Befehlssatz des Treibers natürlich bekannt sein. Die Funktion bleibt sowohl dem Anwender und meist sogar dem Anwendungsprogrammierer verborgen.

Die Gerätedatei enthält nur wenige Informationen. Zuerst enthält sie die Information, welcher Treiber für sie zuständig ist. Die Treiber des Systems sind durchnummeriert. Diese Nummer wird in der Gerätedatei als major number abgelegt. Gerätedateien, die die gleiche major number haben, interagieren mit dem Betriebssystem also über den gleichen Treiber. Entsprechend handelt es sich dabei um ähnliche Geräte. Es ist keineswegs zwingend, dass eine unterschiedliche major number auch ein anderes Gerät bezeichnet. So verwenden viele UNIX-Systeme für jede serielle Schnittstelle mehrere Gerätenummern. Der Unterschied liegt nicht im Gerät, sondern im Treiber. Mit dem einen Treiber wird über die serielle Schnittstelle auf ein Terminal zugegriffen, mit dem anderen Treiber wird die Kommunikation mit einem Modem realisiert, und ein weiterer Treiber könnte beispielsweise Messgeräte steuern, die einen seriellen Anschluss haben.

Da ein Treiber natürlich mehrere Geräte gleicher Bauart bedienen kann, wird eine weitere Kennung benötigt, um diese voneinander zu unterscheiden. Zu diesem Zweck enthält die Gerätedatei noch eine weitere Nummer, die so genannte minor number. Bei einem Zugriff auf die Gerätedatei wird diese Nummer an den Treiber weitergegeben. So ist es für den Treiber erkennbar, welche Platte oder welches Terminal angesprochen werden soll.

Eine Information über den Gerätecharakter wird nicht vom Treiber verborgen, sondern in der Gerätedatei mitgeliefert. Sie gibt an, ob es sich um ein Gerät mit einem blockweisen oder zeichenweisen Zugriff handelt. So ist ein Terminal zeichenorientiert, eine Festplattenpartition oder eine Diskette blockorientiert. Der Hauptunterschied besteht darin, dass ein blockorientiertes Gerät einen Puffer hat, der erst einmal aufgefüllt werden muss. Dann wird der Datenblock als Ganzes an den Treiber geschickt. Blöcke haben beim Datentransfer ein besseres Verhältnis zwischen Nutzdaten und Protokolldaten und sind dadurch viel schneller. Zeichenorientierte Geräte liefern und bekommen ihre Daten byteweise und reagieren darum zeitnah, weil nicht gewartet werden muss, bis ein Block voll ist.

Für manche Geräte existieren beide Arten von Einträgen. Diese Möglichkeit wird eingesetzt, um einerseits einen schnellen Blocktransfer zu gewährleisten, auf der anderen Seite aber einen zeitnahen Zugriff auf die Steuerung des Gerätes zu ermöglichen.

Nicht alle Einträge im Verzeichnis `/dev` sind tatsächlich mit einer Hardware verbunden. Beispielsweise bezeichnen `pty` virtuelle Konsolen, die durch das Starten des Programms `xterm` entstehen.

Eine besondere Variante eines virtuellen Gerätes ist `/dev/null`. Dieses Gerät verwendet man, wenn störende Ausgaben beseitigt werden müssen. Man leitet sie dann dorthin um.

« [Aufgaben eines Treibers](#) | [Hardwarezugriff unter UNIX: `/dev`](#) | [Umgang mit Gerätedateien](#) »

Umgang mit Gerätedateien

Die Informationen der Gerätetreiber kann man durch den Aufruf von `ls -l` anzeigen lassen. Die Liste, die hier erscheint, unterscheidet sich von normalen Dateien. Die Dateigröße wird nicht angezeigt, da Gerätedateien nur einen Verweis auf den Treiber darstellen. Dafür wird die major und die minor number angezeigt. An der ersten Stelle jeder Zeile erscheint ein c oder ein b, je nachdem, ob es ein zeichen- oder blockorientiertes Gerät ist.

In der folgenden Aufstellung sind ein paar Gerätedateien eines Linux-Systems aufgelistet:

crw-rw--	1	root	tty	4,	1	Jul	11	08:38	tty1
crw-rw--	1	root	tty	4,	2	Jul	11	08:38	tty2
crw-rw--	1	root	uucp	5,	65	Sep	24	2001	cua1
crw-rw--	1	root	uucp	5,	66	Sep	24	2001	cua2
brw-rw--	1	root	disk	3,	0	Feb	4	10:43	hda
brw-rw--	1	root	disk	3,	1	Sep	24	2001	hda1
brw-rw--	1	root	disk	3,	2	Jul	10	01:25	hda2
brw-rw--	1	root	disk	8,	0	Sep	24	2001	sda
brw-rw--	1	root	disk	8,	1	Sep	24	2001	sda1
brw-rw--	1	root	disk	8,	2	Sep	24	2001	sda2
brw----	1	arnold	users	2,	0	Mär	27	12:52	fd0

Die ersten vier Geräte sprechen alle serielle Schnittstellen an. Die major number der tty-Geräte ist 4. Obwohl die cua-Geräte ebenfalls die seriellen Schnittstellen ansprechen, haben sie die major number 5, werden also von einem anderen Treiber bedient. Alle vier sind erwartungsgemäß zeichenorientierte Geräte. Darunter befinden sich Platten und Partitionen. Dabei handelt es sich bei hda um die gesamte IDE-Platte und bei hda1 und hda2 um Partitionen. Beides wird vom gleichen Treiber bearbeitet, denn sie haben alle die major number 3. Darunter ist mit sda eine SCSI-Platte aufgeführt. Auf Treiberebene unterscheiden sich SCSI-Platten von IDE-Platten erheblich. Also haben sie auch einen anderen Treiber und eine andere major number.

Das Gerät fd0 ist eine Diskette. Sie ist ebenfalls blockorientiert wie die Festplatten. Interessant an diesem Eintrag ist, dass dieses Gerät dem Benutzer arnold zugeordnet ist, während die anderen Geräte dem Benutzer root zugeordnet sind. Das heißt, dass arnold mit der Diskette nach Belieben umgehen kann. Bei den anderen Geräten ist es so, dass neben dem Eigentümer root auch die Gruppe das Schreib- und das Leserecht hat. Dadurch kann man einem Benutzer den Zugriff ermöglichen, indem man ihn zum Mitglied in der entsprechenden Gruppe macht.

Da eine Gerätedatei keine übliche Datei ist, kann man sie auch nicht mit den normalen Mechanismen für Dateien erzeugen. Zu diesem Zweck gibt es den Befehl `mknod`, der als Parameter alle Informationen benötigt, die eine Gerätedatei ausmachen, also Name, Typ, major und minor number:

```
mknod /dev/tty1 c 4 1
```

Dieser Aufruf erzeugt die Gerätedatei tty1 als zeichenorientiertes Gerät mit dem Treiber mit der major number 4. Das Gerät hat die minor number 1. Das Anlegen einer Gerätedatei ist ein eher seltener Vorgang. Er ist nur dann erforderlich, wenn ein Gerät mit einem speziellen Treiber geliefert wird, der ins System eingebunden werden muss. Aber selbst in solchen Fällen wird vermutlich ein

Installationsskript diese Aufgabe übernehmen.

Interessanter ist da der Aspekt der Datensicherung. Soll oder muss bei einer Datensicherung auch das Verzeichnis /dev gesichert werden, muss das Sicherungstool in der Lage sein, Gerätedateien zu erkennen, zu sichern und später auch per mknod wieder herzustellen.

« [Gerätedateien](#) | [Hardwarezugriff unter UNIX: /dev](#) | [Gerätenamen](#) »

Gerätenamen

Die Namensgebung erscheint anfänglich etwas willkürlich. Aber gewisse Dinge sind über die Systeme hinweg zumindest ähnlich. So erreicht man die Diskette meist mit `/dev/fd0` oder `/dev/fd`. Den Standarddrucker erreicht man bei fast allen Systemen über `/dev/lp`. Einträge wie `modem`, `mouse`, `cdrom`, `diskette` oder `tape` sind sehr sprechend und meist symbolische Links. Wenn sie nicht bereits bei der Installation angelegt worden sind, ist es durchaus sinnvoll, solche Links auf die Standardgeräte zu setzen, um sich nicht ständig kryptische Namen merken zu müssen. Die SCSI-Festplattenlaufwerke werden meist mit `sd` gekennzeichnet. Auf der SUN SPARCstation wird die erste SCSI-Platte am Bus mit `sd0`, die zweite mit `sd1` bezeichnet. Die Partitionen werden dann mit Kleinbuchstaben angesprochen; `sd0a` ist also die erste Partition auf der ersten Platte. Der Aufbau ist also durchaus logisch, aber nicht systemübergreifend. Unter Linux heißt die entsprechende Partition `sda1`, `sda0` gibt es nicht.

Festplatten

Festplatten sind heute in jedem Computer vorhanden. Sie bestehen aus kreisförmigen Metallscheiben, auf die eine magnetische Schicht aufgetragen ist, die die Daten aufnimmt. Da die Magnetisierung erhalten bleibt, wenn der Strom abgeschaltet wird, ist der Speicher permanent. Er bleibt erhalten, bis er wieder geändert wird.

Heutige Festplatten bestehen aus mehreren übereinander angeordneten Scheiben. Für jede Oberfläche gibt es einen eigenen Schreib- und Lesekopf (engl. Head). Die Köpfe sind durch ein Gestänge miteinander verbunden, das durch einen Schrittmotor vom Rand der Platte bis zur Mitte positioniert werden kann. Die Anzahl der Schritte ergeben Spuren (engl. Tracks), die man auch Zylinder nennt. Jede dieser Spuren wird in Sektoren aufgeteilt. Einen Sektor kann man sich vorstellen wie ein Stück einer Torte. Die Position eines Speicherblocks auf einer Platte wird also durch den Kopf, den Zylinder und den Sektor eindeutig beschrieben. Die Anzahl der Köpfe, Zylinder und Sektoren bestimmen die Kapazität. Diese Informationen bezeichnet man auch als die Geometrie einer Platte.

Unterabschnitte

- [SCSI-Platten](#)
 - [IDE-Platten](#)
 - [Inbetriebnahme](#)
 - [RAID-Systeme](#)
 - [RAID 0: Striping](#)
 - [RAID 1: Mirroring](#)
 - [RAID 5](#)
 - [RAID 10](#)
 - [Hardware-RAID](#)
 - [Software-RAID](#)
 - [Überblick über die Einrichtung unter Linux](#)
 - [Partitionieren](#)
 - [Dateisystem erstellen](#)
 - [Swapping](#)
 - [Swapdateien](#)
 - [Einbinden eines Dateisystems: mount](#)
 - [Konsistenz der Dateisysteme](#)
 - [Journal-Dateisysteme](#)
 - [Belegungslisten: df und du](#)
 - [Zuteilung des Plattenplatzes: quota](#)
 - [Maximalwerte](#)
 - [Beispiele](#)
-

SCSI-Platten

Festplatten werden durch einen Controller mit dem Computer verbunden. Dabei kann ein Controller typischerweise mehrere Platten ansteuern. Im UNIX-Bereich sind SCSI-Platten etabliert. Die Platten haben aus Sicht des Controllers keine Geometrie, sondern einfach eine linear aufsteigende Anzahl von Blöcken. Die Umrechnung der einzelnen Blöcke auf die Köpfe, Zylinder und Sektoren übernimmt die Platte selbstständig und entlastet damit den Computer.

Ein SCSI-Controller ist mit einem 8-Bit-Bus und über drei Adressleitungen mit seinen angeschlossenen Einheiten verbunden. Mit drei Adressleitungen lassen sich acht Einheiten adressieren, von denen eine der Controller ist. Die Platten erhalten ihre SCSI-Nummer normalerweise durch das Umstecken von Jumpers. Jedes Gerät am Bus muss eine andere Nummer besitzen.

Wie bei allen schnellen Bussen muss auch der SCSI-Strang an den beiden Enden terminiert werden. Klassischerweise wurde dies durch Aufstecken eines Widerstandsarrays auf die letzten SCSI-Einheiten erreicht. Sicherer ist die Verwendung aktiver Terminatoren. Befindet sich der Controller am Ende des SCSI-Strangs, muss auch er terminiert werden. Bei modernen Controllern ist dies oft durch Controllerbefehle möglich.

Inzwischen hat sich SCSI zu Wide-SCSI weiterentwickelt. Neben dem höheren Takt ist auch eine Verdoppelung des Datenbus auf 16 Bit erfolgt. Auch eine Adressleitung ist hinzugekommen, sodass auch doppelt so viele Geräte an einem Bus angeschlossen werden können. Aus Kompatibilitätsgründen kann der Bus aufgesplittet werden, um Geräte alter Norm zu bedienen. Dabei muss jeder der Teilstränge terminiert werden. Da die Leistung bei solch einer Konstellation nachlässt, empfiehlt es sich, mit zwei Controllern zu arbeiten.

IDE-Platten

Als die PCs erstmalig mit Festplatten ausgestattet wurden, verwaltete das damalige Standardbetriebssystem MS-DOS seine Festplatten noch anhand ihrer Geometrie. Der allererste PC, der IBM 5150, konnte nur 10 MByte große Festplatten einer bestimmten Geometrie bedienen. Der erste Fortschritt war, dass man die Geometrie der Platte konfigurieren konnte und so jede beliebige Platte an den PC anschließen konnte. Inzwischen sind die modernen Platten und Controller in der Lage, die Information über die Geometrie beim Systemstart auszutauschen, sodass sie nicht mehr von Hand eingetragen werden muss.

Auch an den PC kann man mit Hilfe eines zusätzlichen Adapters SCSI-Platten anschließen. Da die Betriebssysteme für den PC erwarten, direkt mit den Köpfen, Zylindern und Sektoren zu arbeiten, täuscht der SCSI-Controller dem PC die Geometrie der angeschlossenen Platten vor. Er rechnet die Koordinaten in Blocknummern um, die er an die Platten sendet, die diese Nummer dann wieder in ihre eigene Geometrie umrechnen. Trotz dieses Umstandes sind SCSI-Platten extrem leistungsstark, aber leider auch teuer.

Inzwischen sind IDE-Platten nicht nur auf dem PC Standard, sondern auch auf dem Macintosh üblich. An einen IDE-Controller lassen sich zwei Platten anschließen. Dabei ist der eine als Master und der zweite als Slave zu konfigurieren. Typischerweise hat ein PC zwei IDE-Controller. Damit lassen sich maximal vier Festplatten anschließen. Da aber auch CD-ROM-Laufwerke an diesen Controller angeschlossen werden, sind drei Festplatten als Maximum eher realistisch.

IDE-Platten sind in der Handhabung und auch in der Geschwindigkeit mit SCSI-Laufwerken in etwa vergleichbar. Allerdings belasten IDE-Laufwerke die CPU intensiver als SCSI. In einem Multitaskingsystem ist dies durchaus relevant.

Inbetriebnahme

Bei einer Neuinstallation des Betriebssystems werden Sie durch das Installationstool geführt, das die einzelnen Schritte in logischer Folge abfragt und manchmal sogar Vorschläge macht. Die Installationstools sind meist auch über die Administrationstools zu erreichen, sodass Sie auf die gleiche Art und Weise auch später zusätzliche Platten vorbereiten können.

Eine Platte ist nach dem Kauf normalerweise leer und vorformatiert. Diese so genannte Lowlevel-Formatierung wurde früher vom Administrator selbst vorgenommen, bevor partitioniert wurde. Inzwischen gibt es keinen Grund mehr, dies selbst zu übernehmen. Der erste Schritt ist also die Partitionierung. Das bedeutet die Aufteilung der Platte in Abschnitte, die noch keine logische innere Struktur haben. Auf Platten, die für den Einsatz unter MS Windows vorbereitet sind, ist häufig bereits eine Partition eingerichtet, die die gesamte Platte umfasst. UNIX-Systeme arbeiten üblicherweise mit mehreren Partitionen, mindestens braucht man eine Swap-Partition und eine Datenpartition.

Um Daten auf der Festplatte ablegen zu können, müssen Strukturen vorhanden sein, die die Dateien und Verzeichnisse aufnehmen können. Diese Struktur wird in einer Partition erzeugt, sodass jede Partition genau ein Dateisystem aufnehmen kann. Unglücklicherweise wird von MS-DOS und auch MS Windows dieser Vorgang ebenfalls Formatierung genannt. Die Formatierung unter MS-DOS entspricht also nicht dem Lowlevel-Formatieren der Platte, sondern dem Erzeugen eines Dateisystems. Es gibt viele verschiedene Dateisysteme, die sogar parallel auf einem Rechner laufen können. So kann Linux eine Datenplatte von MS Windows zur Ablage von Daten verwenden, natürlich mit den Einschränkungen, dass das Dateisystem von MS Windows Groß- und Kleinschreibung nicht unterscheidet und nicht mit allen Dateiattributen umgehen kann.

Wie schon erwähnt wurde, gibt es für jede Partition eine eigene Gerätedatei im Verzeichnis /dev. Einige Systeme wie beispielsweise Linux verwenden diese Gerätedatei auch für den Zugriff auf das darin liegende Dateisystem. Andere wie Solaris und FreeBSD verwenden zwei Gerätedateien, eine für das Dateisystem und eine für die Partition. Dabei erkennt man die Partition an dem zusätzlichen r (für raw device) im Gerätenamen.

Da UNIX alle Dateien in einem einzigen Verzeichnisbaum verwaltet, muss ein Dateisystem an einer bestimmten Stelle in den Verzeichnisbaum eingebunden werden. Dazu wird der Befehl `mount` verwendet. Das Ausklinken des Dateisystems erfolgt mit dem Befehl `umount`. Diese Einbindung bleibt erhalten, solange das System nicht neu gestartet wird. Um Dateisysteme dauerhaft einzubinden, müssen sie in einer Dateisystemtabelle eingetragen werden, die man unter /dev/fstab findet.

Auch Wechselmedien müssen in den Verzeichnisbaum eingebunden und explizit wieder freigegeben werden. Dieser für Benutzer von MS Windows unerklärliche Umstand hat seinen Grund darin, dass so auch Wechselmedien mit Plattenpuffern beschleunigt werden können. Um eine Behandlung der Wechselmedien benutzerfreundlicher zu gestalten, kann man Automount verwenden (siehe S. automount). Bei MacOS X ist dies bereits von vornherein installiert.

Unterabschnitte

- [RAID 0: Striping](#)
 - [RAID 1: Mirroring](#)
 - [RAID 5](#)
 - [RAID 10](#)
 - [Hardware-RAID](#)
 - [Software-RAID](#)
 - [Überblick über die Einrichtung unter Linux](#)
-

RAID-Systeme

RAID-Systeme basieren auf Überlegungen der Autoren Patterson, Gibson und Katz, die sie 1987 in einem Artikel namens »A Case for Redundant Arrays of Inexpensive Disks (RAID)« veröffentlichten.

RAID-Systeme fassen mehrere Platten zu einer logischen Platte zusammen, sodass die laufenden Programme es gar nicht bemerken. Das Ziel des Artikels war es, durch Redundanzen eine höhere Ausfallsicherheit der Platteneinheit zu erreichen. Dabei wurden verschiedene Level, beginnend bei 1, vorgestellt. Später wurde der Level 0 hinzugefügt, obwohl er nichts mit Redundanz zu tun hat. Heute sind hauptsächlich die RAID-Level 0, 1, 5 und schließlich 10 implementiert.

RAID ist eine Technologie, die für Hochverfügbarkeitsrechner unerlässlich ist. Damit verliert ein Plattencrash seinen Schrecken. Diese Systeme sind kein Rundumschutz, sondern verhindern nur den sofortigen Systemstillstand aufgrund der meisten physikalischen Festplattendefekte. Ausfälle durch Anwenderfehler, Programmfehler oder Systemfehler aus anderen Quellen verhindern sie nicht. RAID macht also die Datensicherung keineswegs überflüssig.

RAID 0: Striping

Dieses Verfahren macht aus mehreren physikalischen Platten eine virtuelle. Der offenkundige Vorteil ist, dass man sich nicht um die Verteilung der Verzeichnisse auf die verschiedenen Platten kümmern muss. Dadurch kann es nicht passieren, dass eine Platte vollgeschrieben wird, während die anderen noch massenhaft Platz haben. Dazu kommt, dass es im Extremfall sogar möglich ist, Dateien auf Platten zu speichern, die größer sind als jede einzelne der beteiligten Festplatten.

Auf der anderen Seite steigert das Striping den Durchsatz. Striping bedeutet, dass die Blöcke abwechselnd auf den Platten verteilt sind. Da erwartet wird, dass in vielen Fällen die Dateien sequenziell geschrieben und gelesen werden, wird immer im Wechsel auf die verschiedenen Laufwerke zugegriffen. Im Schnitt wird eine optimale Verteilung der Belastung auf alle Platten erreicht. Dadurch können Parallelisierungsvorteile genutzt werden, die beim sequenziellen Schreiben auf eine einzige Platte nicht erreichbar sind.

RAID 0 hat keinerlei Redundanz, und so ist im Falle des Versagens einer Platte auch der Bereich der anderen Platten betroffen, da die Daten ja nicht auf separaten Dateisystemen liegen.

RAID 1: Mirroring

RAID 1 verwendet mindestens zwei Platten, auf die quasi alle Daten doppelt geschrieben werden. Fällt eine Platte aus, springt die überlebende Platte ein. Die Performance einer solchen Platte ist beim Schreiben geringer als bei normalen Laufwerken, aber auch nicht etwa halb so hoch, da der Controller eine Parallelverarbeitung beim Schreiben erzielen kann. Beim Lesen sollte es keinen Performanceverlust geben.

Als entscheidender Nachteil dieses Verfahrens gilt der hohe Preis pro MByte Speicherkapazität, da doppelt so viele Platten gebraucht werden.

RAID 5

Bei RAID 5 werden mehrere Platten (üblicherweise drei bis fünf) so zusammengefasst, dass der Ausfall einer Platte den Betrieb nicht unterbricht. Das funktioniert auf der Basis von Prüfsummen, die so auf die Platten verteilt werden, dass bei einem Ausfall einer Platte die verbleibenden Platten die Informationen der ausgefallenen errechnen können. Dabei verliert man durch die redundante Speicherung etwas von dem Speicherplatz, den die verwendeten Platten in der Summe bieten würde und eine gewisse Performance, da natürlich das redundante Verteilen der Daten ein wenig Zeit in Anspruch nimmt. Die Lesezugriffe sind aufgrund des Stripings im Verhältnis zu einer einfachen Platte beschleunigt.

RAID 10

RAID 10 ist kein eigenständiger RAID-Level, sondern eigentlich nur die Kombination von RAID 0 mit RAID 1. Es werden also zwei mal zwei Platten per Striping verbunden. Dann werden die Plattenpaare als Spiegelplatten verwendet. Man könnte RAID 10 als beschleunigte Spiegelung ansprechen, denn RAID 10 nutzt die Beschleunigung von RAID 0 und die Redundanz von RAID 1. Im Gegensatz zu RAID 5 ist ein solcher Verbund also schneller, hat aber immer noch den Speicherverlust von 50% durch das Spiegeln.

Hardware-RAID

Zunächst wurde RAID auf Controllerebene, also in Hardware realisiert. An einen Controller werden mehrere Festplatten angeschlossen, die gemeinsam das RAID-System darstellen. Aus Sicht des Betriebssystems ist nicht zu erkennen, dass es sich um mehrere Platten handelt, sondern es betrachtet den Verbund wie eine Platte.

Es gibt zwei Arten, ein RAID-System zu installieren. Die erste Variante ist, einen RAID-Controller zu verwenden, an den mehrere Platten angeschlossen werden. In den meisten Fällen werden das SCSI-Platten sein. Die andere Variante ist, dass man ein externes RAID-System verwendet, das intern einen RAID-Controller besitzt, nach außen aber einfach wie eine gewöhnliche SCSI-Platte angeschlossen wird. Die zweite Variante ist natürlich etwas einfacher in der Handhabung. Die erste Lösung kann daraus Vorteile ziehen, dass sie die Durchsatzvorteile der parallelen Plattenabfragen direkt über den schnellen Systembus und nicht mehr über einen SCSI-Bus quetschen muss.

Hardware-RAID besitzt fast immer auch die Möglichkeit des Hotplugging. Das bedeutet, dass im laufenden Betrieb die Platten gewechselt werden können. Ausgefallene Platten werden meist durch akustische oder Lichtsignale gemeldet und können herausgezogen werden. Eine Ersatzplatte, die man

als kluger Administrator natürlich vorrätig im Schrank liegen hat, kann eingeschoben werden und wird vom Controller wieder ins System eingebunden. Sobald der Controller den Datenbestand der Festplatte wieder aus den verbliebenen Platten restauriert hat, ist das System wieder gegen einen Festplattencrash gesichert.

Software-RAID

Unter Linux wurde RAID auf der Basis des Betriebssystems entwickelt. Es basiert nicht mehr auf ganzen Platten, sondern auf Partitionen. Damit das Striping effektiv ist, müssen die verwendeten Partitionen ähnlich groß sein. Sowohl für das Striping als auch für die Redundanz ist es wichtig, dass die verwendeten Partitionen auf verschiedenen physikalischen Plattenlaufwerken liegen. Das diesbezügliche HOWTO wird noch deutlicher und sagt, dass es »so gut wie immer unsinnig ist, innerhalb eines RAID-Verbundes mehrere Partitionen auf dieselbe Festplatte zu legen«.Linux Software-RAID HOWTO. v0.2.11-2, 5. April 2000. Kap. 3.1. Hardware, letzter Absatz.

Da die Grundlage eines solchen RAID-Verbundes Partitionen sind, ist es irrelevant ist, ob sie auf IDE oder SCSI-Platten liegen. Auch ein Mischbetrieb ist möglich. Allerdings weist das Software-RAID-HOWTO darauf hin, dass IDE-Platten kaum zu parallelisieren sind, selbst wenn sie an verschiedenen Controllern laufen. Performancegewinne wären vor allem bei SCSI-Platten zu erwarten.

Überblick über die Einrichtung unter Linux

Hier soll in Stichworten ausgeführt werden, wie man ein Software-RAID installiert. Dazu muss der Linux-Kernel das Software-RAID unterstützen. In den aktuellen Distributionen ist das normalerweise gewährleistet. Als Beispiel soll ein RAID 0 auf zwei SCSI-Platten installiert werden.

Damit das RAID automatisch beim nächsten Boot aktiviert wird, müssen die Partitionen einen speziellen Typ erhalten, den man mit dem Partitionierungstool `fdisk` setzen kann. Dazu wird `fdisk` für jede Platte gestartet, die eine Partition enthält, die in das RAID eingebunden werden soll. Mit dem Befehl `t` wird der Typ der Partition verändert. Hier muss für die Partition `fd` eingetragen werden. Das ist die Identifikation für »Linux raid auto«:

Anschließend muss in der Konfigurationsdatei `/etc/raidtab` definiert werden, wie das neue RAID auszusehen hat.

raiddev	/dev/md0	
raid-level		0
nr-raid-disks		2
persistent-superblock		1
chunk-size		4

device	/dev/sda1
raid-disk	0
device	/dev/sdb1
raid-disk	1

Der Eintrag `persistent-superblock` erreicht, dass das RAID beim Boot automatisch gestartet werden kann. Die `chunk-size` legt für das Striping fest, wie groß der Datenblock ist, der auf einer Partition gehalten wird. Im Beispiel werden bei einer größeren Datei die ersten 4 KByte auf `sda1`, die nächsten 4 KByte auf `sdb1` geschrieben und so fort. Die Bedeutung der anderen Parameter dürfte offenkundig sein. Nachdem die Datei `raidtab` erstellt ist, kann das RAID mit dem Befehl `mkraid` gestartet werden:

Anschließend kann das RAID wie eine normale Partition über die Gerätedatei /dev/md0 angesprochen werden, und es kann ein Dateisystem auf ihm erstellt werden.

« [Inbetriebnahme](#) | [Festplatten](#) | [Partitionieren](#) »

Partitionieren

Um eine neue Festplatte einsetzen zu können, muss sie zunächst partitioniert werden. Das bedeutet, sie wird in mehrere logische Einheiten untergliedert. Jedes System hat sein eigenes Programm zur Partitionierung einer Platte. Im Allgemeinen findet sich über das jeweilige Systemadministrationstool ein Weg zu dem Programm, das die Partitionierung übernimmt. Man kann sie allerdings auch direkt aufrufen. Unter Linux heißt das Partitionierungstool beispielsweise `fdisk`, unter SCO `divvy` und unter MacOS X `DiskUtility`. Nach der Partitionierung der Platte wird jede Partition über einen Eintrag im Verzeichnis `/dev` referenziert.

Eine regelmäßig gestellte Frage ist die nach der Größe der Partitionen. Ehrliche Antworten werden dabei immer sehr schwammig bleiben. Das liegt daran, dass die Antwort davon abhängt, welches UNIX verwendet wird, ob die Maschine als Workstation oder als Server eingesetzt wird, welche Software installiert werden soll, wie viele Benutzer daran arbeiten und inwieweit diese ihre persönlichen Daten auf der Maschine speichern. Für die Einteilung der Partitionen kann man einige der in Abschnitt , Tuning, genannten Aspekte berücksichtigen. Es ist gut, wenn die Partitionen nicht größer als das Sicherungsmedium sind.

UNIX-Systeme verwenden eine Partition für das Swappen, also das Auslagern von Hauptspeicherbereichen (siehe S. swap). Die Verwendung einer Partition ist wesentlich effizienter als eine Swapdatei, da sie nicht der Fragmentierung des Dateisystems unterliegt. Sie muss allerdings bereits bei der Einteilung der Platte berücksichtigt werden und kann nicht dynamisch wachsen oder schrumpfen.

Auf PCs ist die Partitionierung komplizierter als auf einem klassischen UNIX-System, da auf einer Platte mehrere Betriebssysteme liegen können. Dazu kommt die Problematik, dass es unterschiedliche Arten von Partitionen gibt. An sich sind nur vier so genannte primäre Partitionen pro Platte möglich. Soll auch ein Betriebssystem von Microsoft installiert werden, muss dieses die erste Partition zum Starten bekommen. Werden mehr als vier Partitionen gebraucht, wird eine erweiterte Partition angelegt, in der sich beliebig viele logische Partitionen anlegen lassen. Wichtig für das Starten eines Betriebssystems ist der Master Boot Record (MBR), der sich auf der äußeren Spur einer Festplatte befindet.

Zur Veranschaulichung, wie eine solche Partitionierung auf der Platte aussieht, sehen Sie in Abbildung das Schema einer PC-Platte, die für die Installation von Linux vorbereitet wird.

Die erste Spur enthält den Master Boot Record (MBR). Dann ist auf dieser Platte eine weitere Partition beispielsweise für MS Windows (`hda1`) vorgesehen. Die Partition `hda2` ist eine erweiterte Partition, die mehrere logische Partitionen enthält. In den logischen Partitionen können später die Dateisysteme eingerichtet werden.

Dateisystem erstellen

Während eine Partitionierung eine Aufteilung der Platte ist, ist ein Dateisystem die Rahmenstruktur, in der Verzeichnisse und Dateien abgelegt werden können. Jeder UNIX-Hersteller hat im Laufe der Zeit eigene Dateisysteme entwickelt, die die Geschwindigkeit erhöhen, den Platzverbrauch minimieren und die Sicherheit perfektionieren sollen. Die Entwicklung in diesem Bereich ist noch in vollem Gange, und so muss man damit leben, dass sich die einzelnen Dateisysteme unterscheiden.

Nachdem die Platte partitioniert wurde, wird mit dem Befehl `mkfs` oder `newfs` (Solaris, FreeBSD, MacOS X) ein Dateisystem auf jeder Partition erstellt, das Daten aufnehmen soll. Als Parameter wird das Device erwartet, über das die Partition angesprochen werden kann. Diese Information kann durch das Partitionierungstool ermittelt werden:

```
mkfs /dev/hda3
```

Damit wird ein Dateisystem auf der Partition `/dev/hda3` erzeugt.

Sowohl Solaris als auch MacOS X besitzen zwei Gerätedateien je Partition. Die eine wird als raw device bezeichnet und zeigt auf die Partition. Sie ist an einem `r` im Namen zu erkennen. Unter Solaris werden die Partitionen im Verzeichnis `/dev/rdisk` gesammelt. Diese Gerätedateien werden für die Befehle verwendet, die eine Partition als Parameter verwenden, wie `fsck`, `newfs` und andere. Die anderen Gerätedateien beziehen sich auf die Dateisysteme. Unter Solaris befinden sie sich im Verzeichnis `/dev/dsk`. Sie werden als Parameter für Befehle verwendet, die Dateisysteme bearbeiten, wie beispielsweise `mount`. Andere Systeme wie auch Linux kennen diese Unterscheidung nicht. Die Befehle verwenden immer die Gerätedateien der Partitionen und »wissen« selbst, ob die Partition oder das in ihr befindliche Dateisystem gemeint ist.

`mkfs` erzeugt das Standarddateisystem. Soll ein anderes als dieses verwendet werden, wird per Option die Kennung für das Dateisystem angegeben. Soll beispielsweise unter Linux ein MINIX-Dateisystem erzeugt werden, bekäme `mkfs` die Option `-t minix`. Letztlich ruft das Programm dann das auf dieses Dateisystem spezialisierte Tool auf, in diesem Fall `mkfs.minix`.

In alten Versionen von `mkfs` musste man noch angeben, wie viele Blöcke das Dateisystem haben sollte und wie groß die Blöcke sein sollten. Heutige Versionen von `mkfs` errechnen die Anzahl der Blöcke aus der Größe der Partition selbst. Auch die Blockgröße ist vorgegeben. Früher war das ein gewisser Tuningparameter. Je größer die Blöcke sind, desto schneller wird der Zugriff auf große Dateien. Allerdings wird bei vielen kleinen Dateien auch der Platzverbrauch erheblich ansteigen. Die Blockgröße spielt bei heutigen Dateisystemen keine entscheidende Rolle mehr, da die Systeme mehrere kleine Dateien in einen Block schreiben können.

Unterabschnitte

- [Swapdateien](#)

Swapping

Neben den Datenpartitionen verwendet UNIX eine Partition zum Swappen. Unter Swapping versteht man das Ein- und Auslagern von Prozessen aus dem Hauptspeicher auf die Festplatte. UNIX lagert Prozesse, die längere Zeit nicht aktiv wurden aus, wenn es im Hauptspeicher eng wird. Dadurch ist es möglich, dass mehr Prozesse gestartet sind, als in den Hauptspeicher passen.

Nun ist das Auslagern auf Platte und das Zurückholen in den Hauptspeicher ein recht zeitaufwändiges Verfahren, da die Platte einen sehr viel langsameren Zugriff hat als Hauptspeicher. Tatsächlich belastet das Swappen das System beim ersten Auftreten von Speicherengpässen erheblich. Je länger das System allerdings läuft, desto ruhiger wird es, weil mit der Zeit diejenigen Hintergrundprozesse ausgelagert werden, die selten zum Einsatz kommen. Solche Hintergrundprozesse, die nur auf ein bestimmtes Ereignis warten, um aktiv zu werden und die restliche Zeit schlafen, gibt es in einem UNIX-System in großer Zahl. Insofern ist es auch nicht beunruhigend, wenn der Swapbereich belegt ist.

Kritisch wird es erst dann, wenn Bewegung ins Spiel kommt, wenn also das System häufig auslagern muss. Dann sollte man den Hauptspeicher erweitern, weil das ständige Auslagern auf Platte die Maschine belastet. Wenn der Speicher derart überlastet ist, dass die Maschine ständig die Prozesse ein- und auslagert und sonst fast nicht mehr arbeitet, redet man vom Thrashing.

An sich ist die Bezeichnung Swappen nicht mehr ganz korrekt, da es an sich das Auslagern ganzer Prozesse bezeichnet. Heutige Systeme arbeiten längst mit Pagingverfahren. Das bedeutet, dass das Auslagern nicht mehr auf Prozessebene stattfindet, sondern dass Speicherseiten fester Größe ausgelagert werden. Man kann sich das so vorstellen, dass der Hauptspeicher in Kacheln gleicher Größe aufgeteilt ist und das System kontrolliert, welche Kachel wie oft benutzt wird. Auf diese Weise können Prozesse teils im RAM und teils auf der Platte liegen. Das Paging ist aufwändiger, aber auch wesentlich effizienter als das Swapping. Man spricht auch von virtuellem Speicher, da der Prozess nicht merkt, dass sich sein Speicher zum Teil eigentlich auf der Platte befindet.

Nachdem die Partition für das Swappen festgelegt ist, muss beispielsweise bei Linux analog zum Anlegen eines Dateisystems auch die Swap-Partition initialisiert werden. Dazu dient der Befehl `mkswap`. Als Parameter benötigt er die Gerätedatei der Partition.

Das Swappen wird beim Systemstart in einem rc-Skript gestartet. Der Befehl zum Aktivieren lautet `swapon`. Dabei wird es üblicherweise mit der Option `-a` gestartet. Das führt dazu, dass in der `/etc/fstab` nach den Swapdevices geschaut wird und diese aktiviert werden.

Über die Größe der Swap-Partition gibt es unterschiedliche Aussagen. Die Faustregel, den Hauptspeicher noch einmal als Swap zu nehmen, stammt wohl daher, dass das System im Falle eines Kernel-Panic, also eines Totalzusammenbruchs des Betriebssystems, einen Speicherabzug in die Swap-Partition schreibt. Abgesehen von dieser Überlegung ist die Regel wenig sinnvoll, da die Summe

aus Swap-Partition und RAM den virtuellen Speicher ausmachen. Wenn also das RAM bereits knapp ist, sollte der Swapbereich großzügig sein. Ist jede Menge RAM im System, kann man vielleicht sogar ganz ohne Swap-Partition auskommen und nur als Sicherheit noch eine Swapdatei anlegen. Die Summe aus RAM und Swapbereich sollte auch im schlimmsten Fall noch ausreichend sein.

Fast jedes UNIX verwendet eine Swap-Partition und keine Datei. Der Vorteil liegt in der Geschwindigkeit. Der Zugriff auf die Swap-Partition erfolgt wesentlich schneller als der Zugriff über das Dateisystem. Der Nachteil ist, dass man zum Installationszeitpunkt festlegen muß, welche Größe der Swapbereich benötigen wird.

Swapdateien

Einige UNIX-Systeme können auch mit Swapdateien arbeiten. Dies kann eine schnelle Lösung sein, wenn der Swapbereich zu eng wird und eine erneute Partitionierung vermieden werden soll. Es ist aber vor allem eine dynamische Lösung, die kurzfristige, extreme Spitzen abfängt. Installiert man auf einem System zusätzlich zur Swap-Partition eine Swapdatei, wird es auch dann keinen Programmabbruch wegen zu wenig Speicher geben, wenn plötzlich mehr Prozesse gestartet werden, als je voraussehbar war.

Unter HP-UX kann am einfachsten über das Administrationstool sam eine Swapdatei eingerichtet werden. Dabei entsteht folgender Eintrag in der /etc/fstab (siehe dazu S. fstab):

```
/dev/vg00/lvol8 /users hfs rw,suid 0 2
/dev/vg00/lvol8 /users swapfs min=3840,lim=12800,pri=2 0 2
```

In der oberen Zeile sieht man, dass das Plattendevise lvol8 ein normales hfs-Dateisystem ist. Es wird auch tatsächlich für Benutzerdaten verwendet. Daneben liegt aber auch im Verzeichnis /users das so genannte Dateisystempaging. Betrachtet man den Inhalt von /users, findet man dort ein Verzeichnis namens paging, in dem mehrere Dateien liegen, die zum Swappen verwendet werden. Die Parameter geben übrigens an, dass mindestens 3840 Blöcke verwendet werden sollen. Das Limit (lim=) beträgt 12.800 Blöcke. Alle installierten Swapbereiche bekommen eine Priorität. Bei der Anforderung von virtuellem Speicher wird mit der geringsten Priorität begonnen. Da auf diesem System auch eine Swap-Partition mit der Priorität 1 ist, wird zuerst diese ausgeschöpft, bevor auf die Datei zugegriffen wird.

« [Dateisystem erstellen](#) | [Festplatten](#) | [Einbinden eines Dateisystems: mount](#) »

Einbinden eines Dateisystems: mount

Nachdem ein Dateisystem erzeugt wurde, muss es in den Verzeichnisbaum eingehängt werden, damit es nutzbar ist. Der Aufruf zum Einhängen eines Dateisystems lautet `mount`, der zum Freigeben `umount`.

Bei einem neuen System sollte vor der Installation klar sein, welches Dateisystem an welche Stelle im Verzeichnisbaum eingehängt werden soll. Bei der Installation wird dies oft automatisch vom Installationsprogramm unterstützt. Soll beispielsweise eine eigene Partition das Verzeichnis `/usr` aufnehmen, wird das Verzeichnis angelegt und das Dateisystem mit einem `mount` eingehängt. Intern werden also quasi folgende Befehle ausgeführt:

```
mkdir /usr
mount /dev/sda2 /usr
```

Wird dagegen nachträglich eine Platte als Speichererweiterung eingebaut, wird man sie unter einem Dummy-Verzeichnis in den Verzeichnisbaum einhängen und die Zugriffe über einen symbolischen Link realisieren. Manchmal werden Platten über die Verzeichnisse `/u`, `/u1` oder `/v` eingehängt. Etwas ordentlicher ist es, wenn man innerhalb eines Verzeichnisses `/mount` oder `/drives` Verzeichnisse für das Einhängen der Platten einrichtet. Durch einen symbolischen Link kann man anschließend jedes beliebige Verzeichnis auf die Platte verlegen. Zum Beispiel:

```
mkdir /mount/hda3
mount /dev/hda3 /mount/hda3
ln -s /mount/hda3 /home/tacoss
```

Das Verzeichnis `/mnt` eignet sich dazu weniger, da es traditionell gern verwendet wird, um kurzfristig Dateisysteme einzuhängen.

Nachdem die neue Platte eingehängt worden ist, muss man normalerweise einige Verzeichnisbäume auf die neue Platte schaffen, um auf den vollen Dateisystemen wieder Luft zu bekommen. Beim Umschichten der Dateien und Verzeichnisse ist es wichtig, dass nicht nur alle Daten transportiert werden, sondern auch, dass alle Dateirechte erhalten bleiben. Um das zu gewährleisten, sollte die Verzeichniskopie per `tar` (siehe S. `tarcopy`) erfolgen, damit die Eigenschaften der Dateien erhalten bleiben.

In moderneren Systemen kennt der Befehl `cp` auch die Option `-R` zum Kopieren kompletter Verzeichnisse und die Option `-p` zum Erhalt der Dateieigenschaften. Falls symbolische Links in dem Verzeichnis existieren, muss auch kontrolliert werden, ob `cp` auch mit ihnen richtig umgeht.

Schließlich wird ein symbolischer Link anstelle des ursprünglichen Verzeichnisortes erzeugt, der auf die neue Position auf der neuen Platte zeigt. Danach ist für die normalen Anwendungsprogramme kein Unterschied zu vorher zu erkennen, außer dass nun wieder Platz zum Arbeiten ist.

Damit das System beim nächsten Booten die Platte wieder an der richtigen Stelle einhängt, wird ein Eintrag in der Datei `/etc/fstab` für die Platte eingerichtet. Die Zeile für das obige Beispiel würde unter

Linux so aussehen:

/dev/hda3	/mount/hda3	ext2	defaults 1 1
-----------	-------------	------	--------------

Die Datei /etc/fstab enthält alle Dateisysteme, die beim Booten bereits eingebunden werden. Eine typische fstab sieht so aus:

/dev/hda3	swap	swap	defaults 0 2
/dev/hda2	/boot	ext2	defaults 1 2
/dev/hda5	/	ext2	defaults 1 1
/dev/cdrom	/cdrom	auto	ro,noauto,user,exec 0 0
/dev/fd0	/floppy	auto	noauto,user 0 0

In der ersten Spalte stehen die Partitionen. Die zweite Spalte bezeichnet den Ort, wo die Partition ihren Platz im Verzeichnisbaum findet. Die dritte Spalte bezeichnet den Typ des Dateisystems. ext2 ist beispielsweise das Standarddateisystem von Linux. Die vierte Spalte enthält Optionen, die kommasepariert aufgeführt werden, aber aus nahe liegenden Gründen kein Leerzeichen enthalten dürfen. Diese Optionen entsprechen denen des Befehls mount. ro bezeichnet schreibgeschützte Systeme, wie beim CD-ROM-Laufwerk zu erwarten. user bedeutet, dass der Anwender das Dateisystem mounten darf und anschließend auch Zugriffsrechte auf dieses Dateisystem hat. noauto verhindert, dass die CD-ROM bereits beim Booten in das Dateisystem eingehängt wird. Die Zahl in der fünften Spalte ist 1, wenn das Dateisystem bei einem dump (siehe S. dump) berücksichtigt werden soll. Die Zahl in der sechsten Spalte wird von fsck verwendet, um festzustellen, in welcher Reihenfolge die Dateisysteme beim Booten geprüft werden müssen. Steht dort eine 0, braucht das System nicht geprüft zu werden.

Allein durch die unterschiedlichen Namen der Dateisysteme ist die Datei fstab nicht zwischen den UNIX-Systemen portabel. Da die Partitionseinteilung aber auch sehr individuell für jeden Computer ist, muss sie auch nicht zwischen den Systemen austauschbar sein. Immerhin sind die Varianten so konsistent, dass man sich auch unter einem fremden UNIX-Derivat sofort darin zurecht findet.

Um den Spezialisten für die einzelnen Systeme ein gewisses Spezialwissen zu garantieren, haben die Hersteller für die gleiche Datei verschiedene Namen in Umlauf gebracht:

[Namensvarianten für fstab]C|C System & Name der Datei

SCO & /etc/default/filesys

HP-UX 10.10 & /etc/fstab

Solaris & /etc/vfstab

Ein Dateisystem, das in den Verzeichnisbaum eingehängt wurde, muss vor dem physikalischen Entfernen des Mediums wieder ausgehängt werden. Der Befehl dazu lautet umountEs darf kein n hinter dem u stehen.. Wechselmedien kann man erst entnehmen, wenn sie ausgehängt sind. Auch wenn ein Dateisystem per fsck geprüft werden soll, muss es erst ausgehängt werden.

Ein Dateisystem kann nur dann aus dem Verzeichnisbaum genommen werden, wenn es von niemandem mehr benutzt wird. Schlägt der Befehl umount fehl, gibt es noch Benutzer, die das Dateisystem verwenden. Wenn Sie den Befehl fuser auf das Wurzelverzeichnis des Dateisystems anwenden, ist schnell ermittelt, welche Prozess-ID die Platte in Beschlag nimmt.

Bestimmte Systemverzeichnisse sind natürlich immer in Gebrauch, wie etwa das Verzeichnis /var, das

von beinahe jedem Hintergrundprozess verwendet wird. Solche Verzeichnisse kann man nur im Single-User-Modus aushängen.

« [Swapping](#) | **Festplatten** | [Konsistenz der Dateisysteme](#) »

Konsistenz der Dateisysteme

UNIX achtet von sich aus auf die Konsistenz seiner Dateisysteme. Bei jedem Boot wird geprüft, ob das System beim letzten Mal korrekt heruntergefahren wurde. War dies nicht der Fall, erfolgt zwingend eine Prüfung des Dateisystems. Aber selbst, wenn alles in Ordnung ist, wird nach einer gewissen Zahl von Startvorgängen eine Zwangsprüfung durchgeführt.

Werden bei dieser Prüfung erhebliche Mängel festgestellt, stoppt der Bootprozess mit einer Meldung. Auf der Konsole erscheint ein Prompt mit der Aufforderung, die Platte zu renovieren, deren Gerätebezeichnung ebenfalls angezeigt wird. Zu diesem Zeitpunkt befindet sich das System im Single-User-Modus. Wenn Sie nicht gerade besondere Kenntnisse über den internen Aufbau des Dateisystems besitzen, können Sie lediglich `fsck` für das Dateisystem noch einmal starten. Da dieser Lauf im Dialog stattfindet, kann das System nun fragen, was es beispielsweise mit defekten Sektoren machen soll. Ihnen wird nichts anderes übrig bleiben, als die Vorschläge von `fsck` zu akzeptieren. Sie können sich immerhin die gemeldeten Fehler notieren und haben damit einen Eindruck, wie groß die Schäden sind und ob doch vielleicht eine Datenrücksicherung erforderlich ist. Um auf einem Linux-System die dritte Partition zu prüfen und ggf. zu reparieren, verwenden Sie den folgenden Befehl:

```
fsck /dev/hda3
```

Sektoren, auf die durch die Reparatur oder durch den Schaden nicht mehr zugegriffen werden kann, werden in das Verzeichnis `lost+found` geschoben, das auf jeder Partition direkt unter ihrem Wurzelverzeichnis vorhanden ist.

Hat man Anlass zu der Vermutung, dass ein Dateisystem nicht konsistent ist, kann man den `fsck` auch im laufenden Betrieb starten. Allerdings sollte das bei einem mit `umount` abgehängten Dateisystem durchgeführt werden, da vermieden werden muss, dass andere Benutzer parallel darauf zugreifen. Ist ein Dateisystem betroffen, das man nicht im laufenden Betrieb aushängen kann, muss man in den Single-User-Modus wechseln (siehe S. `singleuser`) und von dort den `fsck` starten.

Journal-Dateisysteme

Die meisten Systeme bieten inzwischen das Journaling. Die englische Originalbezeichnung für Journal-Dateisysteme lautet journaling filesystem. auf den Dateisystemen an. Das bedeutet, dass Dateisystemänderungen in Protokolldateien abgelegt werden. Sie werden dann sukzessive auf das Dateisystem übertragen, bis ein konsistenter Zustand wieder hergestellt ist. Erst dann wird der Eintrag aus der Protokolldatei wieder gelöscht. Dadurch ist es auch im Falle eines Absturzes oder Stromausfalls möglich, einen sicheren Dateisystemzustand binnen weniger Sekunden wieder herzustellen. Da nicht das komplette Dateisystem überprüft werden muss, kann fsck schnell ein einwandfreies System wieder herstellen.

Die Wirkungsweise eines Journal-Dateisystems garantiert eine Konsistenz der Dateisystemstruktur, sofern keine Fehler in der Software vorliegen. Allerdings garantiert es keine Unverwundbarkeit Ihrer Daten. Ist der letzte Zustand vor einem Absturz nicht mehr herstellbar, weil er sonst eine Inkonsistenz des Dateisystems herbeiführen würde, wird ein älterer Stand verwendet. Es können also durchaus Daten verloren gehen.

Solaris ab Version 7 bietet optional die Möglichkeit, das eigene Dateisystem UFS mit einem Journaling-Mechanismus zu versehen. Dazu muss dem Befehl mount oder dem Eintrag in der Datei /etc/vfstab lediglich die Option -o logging mitgegeben werden. vgl. Nemeth, Evi / Snyder, Garth / Seebass, Scott / Hein, Trent R.: UNIX Systemverwaltung. Markt+Technik - Prentice Hall, 2001. S. 206.

HP-UX kann ab Version 10.20 das Dateisystem VXFS, das Veritas Dateisystem, verwenden, das auch Journaling zur Verfügung stellt. Man legt ein solches Dateisystem mit dem Parameter -F vxfs zum Befehl newfs an. vgl. Nemeth, Evi / Snyder, Garth / Seebass, Scott / Hein, Trent R.: UNIX Systemverwaltung. Markt+Technik - Prentice Hall, 2001. S. 210.

SGI hat sein Journal-Dateisystem XFS inzwischen als Open Source zur Verfügung gestellt, sodass es auch unter Linux zur einsetzbar ist. Dort stehen auch das Reiser-Dateisystem oder ext3 zur Verfügung.

Die Verwendung von Journal-Dateisystemen bedeutet keinen spürbaren Verlust an Performance. Durch die redundante Datenhaltung geht im geringen Umfang Plattenspeicher verloren. vgl. Milz, Harald: Crashfest - SGI XFS auf SuSE 7.1.. Linux-Magazin 07/2001, S. 86-89.

Einige der Dateisysteme sind noch recht neu. Da das Dateisystem eine zentrale Rolle bei der Betriebssicherheit eines Rechners spielt, sollte man sehr genau die Fehlerhinweise in der Dokumentation und auch die Hinweise der Hersteller beachten, bevor man seine Daten einem neuen Dateisystem anvertraut. So gab es eine durchaus engagiert geführte Debatte über die Einsetzbarkeit des Reiser-Dateisystems. vgl. <http://www.dignatz.de/linux-xp222>

Belegungslisten: df und du

Der Befehl `df` (disk free) zeigt eine Liste aller Dateisysteme mit deren Platzverbrauch und ihrem Füllgrad in Prozent. Der Befehl ist auch nützlich, um zu sehen, welche Dateisysteme wo eingehängt sind. Man kann sich auch einzelne Dateisysteme ansehen, indem man das Dateisystem als Parameter angibt.

```
gaston> df
Dateisystem      1k-Blöcke   Benutzt verfügbar Ben% montiert auf
/dev/hda5        7889920   4150752   3338348   56% /
/dev/hda2         225517     3578    210292    2% /boot
/dev/hda7        6324896   3177260   2826340   53% /home
gaston>
```

Ältere Systeme liefern diese Angaben in 512-Byte-Blöcken statt in Kilobyte. Oft stellt dann das Flag `-k` auf Kilobyte-Anzeige um. Auch der Füllgrad in Prozent wird nicht von allen Systemen mit angezeigt. Durch die Option `-v` erscheint eine vollständige Auflistung.

Der Befehl `du` (disk used) zeigt die belegten Blöcke der als Argument angegebenen Dateien bzw. Verzeichnisse. Auch hier zeigen ältere Systeme manchmal die Größen in 512-Byte-Blöcken an. In den Manpages ist das dokumentiert. Im folgenden Beispiel wurde `du` im Verzeichnis `/var/log` durchgeführt.

```
gaston# du
40      ./news/OLD
176     ./news
428     ./httpd
4       ./uucp
116     ./cups
4       ./vbox
172     ./samba
17728   .
gaston#
```

Da `du` alle Verzeichnisse rekursiv durchläuft, bekommt man eine recht lange Liste. Durch die Option `-s` werden nur die angeforderten Verzeichnisse aufgelistet. Dennoch muss natürlich der Verzeichnisbaum komplett durchlaufen und aufsummiert werden, was eine gewisse Belastung der Maschine darstellen kann. Wer schon immer eine Liste mit der Platzverteilung auf dem Produktionsserver haben wollte, sollte also seine Neugier nicht unbedingt um 11 Uhr vormittags befriedigen, sondern die Anfrage als `at-Job` (siehe S. [at](#)) in die Nacht verlegen.

```
gaston# du -s
17728   .
gaston#
```

Zuteilung des Plattenplatzes: quota

Quota dienen dazu, jedem Anwender und jeder Gruppe einen bestimmten Plattenspeicher zuzuteilen, der nicht überschritten werden darf. So kann man verhindern, dass eine Aushilfskraft Musikdateien aus dem Internet sammelt und die Finanzbuchhaltung zum Absturz bringt, weil für sie kein Platz mehr auf der Platte ist. Unabhängig von solchen Szenarien gibt es bei gesetzten Quota den psychologischen Effekt, dass die Benutzer merken, dass Plattenspeicher endlich ist. In den meisten Fällen werden sie einmal ihre Daten durchsortieren und überholte Daten löschen, bevor sie beim Administrator um weiteren Speicher bitten. Das wiederum entlastet auch die Datensicherung.

Mit drei Parametern wird eine Quotagrenze bestimmt: Softlimit, Hardlimit und Gnadenfrist (grace period). Das Softlimit darf zwar überschritten werden, aber nur für eine gewisse Zeit, die als Gnadenfrist bezeichnet wird. Das Hardlimit ist die Grenze, die der Benutzer nicht überschreiten darf. Alle drei Parameter gibt es zweimal, jeweils für die Anzahl der Blöcke und die Anzahl der Dateien. Genauer: die Anzahl der i-nodes.

Zur Installation muss der Kernel die Fähigkeit haben, mit Quota umzugehen. Das kann ggf. ein Neubilden des Kernels erforderlich machen.

In der Datei `/etc/fstab` wird hinterlegt, welche Dateisysteme durch Quota überwacht werden:

<code>/dev/hda3</code>	<code>/mount/hda3</code>	<code>ext2</code>	<code>defaults,usrquota</code>	<code>1</code>	<code>1</code>
------------------------	--------------------------	-------------------	--------------------------------	----------------	----------------

In das Wurzelverzeichnis des Dateisystems müssen zwei Dateien gelegt werden. Im folgenden Beispiel ist das Wurzelverzeichnis `/mount/hda3`. Die Dateien heißen `quota.user` und `quota.group`. Letztere muss auch dann angelegt werden, wenn, wie in diesem Beispiel, gar keine Groupquota angelegt werden.

```
touch /mount/hda3/quota.user
touch /mount/hda3/quota.group
chmod 600 /mount/hda3/quota.user
chmod 600 /mount/hda3/quota.group
mount /dev/hda3 /mount/hda3 -o remount
```

Der Befehl `touch` legt eine leere Datei an, wenn noch keine Datei vorhanden ist. Anschließend werden die Rechte der Dateien auf 600 gesetzt. Das ist zwingend, damit nur root diese Dateien lesen und schreiben kann. Ansonsten arbeitet das Quota-System nicht. Zum Schluss wird ein Wiedereinhängen des Dateisystems gestartet, damit der Eintrag in der `fstab` gelesen wird.

Nun wird ermittelt, welche Benutzer wie viel Platz auf der Platte belegt haben. Dazu gibt es das Programm `quotacheck`:

```
quotacheck -avug /dev/hda3
```

Dieser Vorgang dauert eine Weile. Die `quota.user` wird mit Daten gefüllt. Anschließend kann man die Limits für die Benutzer einstellen.

```
edquota -u arnold
```

edquota startet den Standardeditor (meist vi) mit folgenden Einträgen:

```
Quotas for user arnold:
/dev/hda3: blocks in use: 8117, limits (soft=0, hard=0)
          inodes in use: 470, limits (soft=0, hard=0)
```

Durch eine Änderung der Werte in den Klammern werden das Soft- und das Hardlimit des Benutzers für die jeweilige Platte eingestellt.

edquota -t zur Einstellung der Gnadenfrist betrifft alle Benutzer gemeinsam:

```
Time units may be: days, hours, minutes, or seconds
Grace period before enforcing soft limits for users:
/dev/hda3: block grace period: 7 day, file grace period: 7 days
```

Mit `quotaon -a` wird die Überwachung für alle gestartet, mit `quotaoff -a` wird sie wieder abgeschaltet. Soll die Überwachung auch nach einem Reboot erfolgen, muss der Befehl in einer rc-Datei eingetragen werden.

Ist das Quotasystem aktiv, erhält der Benutzer bei Überschreitung seines Limits eine Meldung am Bildschirm, und der Vorgang wird abgebrochen.

Die ständige Überprüfung aller Dateizugriffe darauf, ob eine Grenze überschritten wird, ist natürlich nicht kostenlos. Quota belasten die Performance der Dateizugriffe. Ist die Zugriffsgeschwindigkeit ein wichtiges Kriterium, kann man durch Partitionieren erreichen, dass Benutzer nicht den wichtigen Anwendungen den Plattenplatz wegnehmen. Man braucht nur dafür zu sorgen, dass die betriebswichtigen Anwendungen ihre Daten auf einer eigenen Partition ablegen. Administrieren Sie eine Maschine mit vielen Benutzern, die sich als Sammler und Jäger im Internet betätigen und neben Bildern, mp3-Dateien nun auch digitale Filme entdecken, führt an der Einrichtung von Quota langfristig kein Weg vorbei.

« [Belegungslisten: df und du](#) | [Festplatten](#) | [Maximalwerte](#) »

Unterabschnitte

- Beispiele

Maximalwerte

Jedes Computersystem hat seine Limits, auch UNIX. Diese Grenzen verschieben sich immer wieder, sind aber dennoch vorhanden. Diese Grenzen können die Anzahl der offenen Dateien, die maximale Anzahl von Benutzern, die Größe von Dateien oder Partitionen betreffen. Jedes Mal, wenn ein System eine solche Grenze erreicht, hat das in der Praxis unangenehme Folgen. Dazu kommt, dass solche Grenzen oft unerwartet erreicht werden. So haben beispielsweise Zusammenbrüche, die in der Zeit zwischen 10 und 11 Uhr vormittags gehäuft vorkommen, oftmals ihren Grund darin, dass die Limits für Benutzer, Prozesse oder offene Dateien erreicht werden. Dies ist nämlich erfahrungsgemäß die Zeit, in der die meisten Anwender mit dem Zentralcomputer arbeiten.

Es ist wichtig, die Grenzen seines Systems zu kennen. Ein kluger Administrator wird aufmerksam, wenn in der Systemdokumentation irgendwelche Limits genannt werden. Eine Quelle für das Auffinden solcher Grenzen sind die Kernelparameter, die Sie sich vom Administrationstool anzeigen lassen können. Diese Werte sind für die normale Einsatzumgebung voreingestellt. Es kann für Ihr Einsatzgebiet ein Parameter zu gering eingestellt sein. Dabei ist es wenig sinnvoll, die Parameter wahllos nach oben zu setzen. Dadurch steigt der Ressourcenverbrauch vermutlich immens, ohne dass einer der Benutzer etwas davon hat.

Neben den durch den Administrator festgesetzten Grenzen ergeben sich Limits durch Programme oder Kapazitätsüberschreitungen von Zählern oder Verweisen. Solche Grenzen treten gehäuft bei allen Zweierpotenzen auf, deren Exponent durch 8 teilbar ist, also bei $2^8 = 256$, $2^{16} = 65.536$, $2^{24} = 16.777.216$ oder $2^{32} = 4.294.967.296$. Auch die Hälfte dieser Werte ist relevant, weil in C und anderen Programmiersprachen ein Bit für das Vorzeichen gebraucht wird. Lange Zeit war 2 GByte, also die Hälfte von 2^{32} , die Grenze für die maximale Größe einer Datei und die maximale Größe eines Dateisystems. Heute erlauben fast alle UNIX-Derivate größere Dateisysteme. Dies kann auch recht bedenkenlos genutzt werden.

Bei der Größe der Datei spielt aber neben dem Betriebssystem auch das Programm eine wesentliche Rolle, das die Datei liest und schreibt. Schreibt das Programm sequenziell in die Datei, wie das beispielsweise bei Protokolldateien der Fall ist, bestimmt das Betriebssystem die maximale Größe einer Datei. Wird aber im direkten Zugriff gearbeitet, dann verwendet das Programm den Aufruf `lseek()` (siehe S. `lseek`) mit einem 32-Bit-Wert als Positionierparameter. Mit 32 Bit können maximal 4.294.967.296 Bytes adressiert werden. Sobald aber ein Vorzeichen verwendet wird, verliert man ein Bit und die maximale Größe beträgt 2.147.483.648, also 2 GByte. Programmtechnisch ist das Problem zu umgehen, indem nicht vom Anfang der Datei aus positioniert wird. Dann muss das Programm bereits mit der Absicht geschrieben sein, die 2-GByte-Grenze zu durchbrechen. Das ist dann im Allgemeinen auch dokumentiert.

Beispiele

SCO hat noch eine zusätzliche Grenze, die in den Kernelparametern durch ULIMIT festgelegt wird. In einer Standardinstallation ist die maximale Dateigröße auf 1 GByte beschränkt. Den Parameter kann man ändern, indem man `scoadmin` startet. Darin findet man den Hardware/Kernel Manager. Als Option wird »Tune Parameter« angeboten. Unter der Gruppe »User and Group configuration« findet man zuerst der Parameter `NOFILES`, der die Anzahl der Dateien pro Prozess bestimmt, und als zweites ULIMIT. Nach der Änderung muss unter »Hardware/Kernel Manager« noch die Option »Relink Kernel« aufgerufen werden und der Parameter ist korrigiert.

Damit nicht genug: ULIMIT wird auch noch als Environmentvariable in der Datei `/etc/default/login` eingesetzt, die vom Terminal gestartete Prozesse auf diese Größe beschränkt. Der Wert, den ULIMIT angibt, ist die maximale Anzahl der Blöcke, die 512 Byte groß sind. Also sind 2 GByte 4.194.303 Blöcke.

Eine weitere Schranke kann die Anzahl der maximal geöffneten Dateien sein. Dabei gibt es zwei Werte. Der eine beschränkt die Gesamtzahl der offenen Dateien und der andere die Anzahl der Dateien, die ein einzelner Prozess öffnen darf.

Bei einer Linux-Standardinstallation wird die Anzahl der gleichzeitig offenen Dateien auf 1024 begrenzt. Die Einstellung befindet sich in der Datei `/usr/src/linux/include/linux/fs.h` und heißt `NR_FILES`. `NR_INODES` ist um den gleichen Faktor zu erhöhen. Eine Neubildung des Kernels ist notwendig, um den Parameter zu erhöhen. Dieser Wert scheint zunächst hoch. Allerdings kann in einer Serverumgebung mit 200 Benutzern jeder Anwender nur noch fünf Dateien öffnen.

Ansonsten gibt es bei älteren Systemen manchmal eine Begrenzung der Plattenkapazität. So konnte beispielsweise HP-UX bis zur Version 9 nur Platten von maximal 4 GByte Größe verwalten.

« [Zuteilung des Plattenplatzes: quota](#) | [Festplatten](#) | [Datensicherung](#) »

Datensicherung

Die Datensicherung ist die wichtigste Aufgabe des Administrators. Die Verantwortung dafür kann ihm niemand abnehmen. Wenn die Datensicherung über einen längeren Zeitraum nicht klappt, ist das Unternehmen beim nächsten Plattencrash so gut wie pleite.

Unterabschnitte

- [Vorüberlegungen](#)
 - [Zeitpunkt der Sicherung](#)
 - [Inkrementelle Sicherung](#)
 - [Sicherheit der Sicherung](#)
 - [Wiederherstellung ausprobieren](#)
- [Das Bandlaufwerk](#)
 - [Steuerung eines Bandlaufwerkes: mt](#)
- [dump](#)
 - [Daten zurück: restore](#)
 - [Beispiel für eine Fernsicherung mit dump](#)
- [tar \(tape archiver\)](#)
 - [Zusammensetzung des tar-Kommandos](#)
 - [Die Steuerungsdatei /etc/default/tar](#)
- [cpio](#)
 - [Sichern, anschauen und zurückholen](#)
 - [Wichtige Optionen](#)
 - [Verzeichniskopie via cpio](#)
- [Medien kopieren: dd](#)
- [Andere Sicherungstools: AMANDA](#)
- [Beispiel für eine Sicherung auf CD-RW](#)

Unterabschnitte

- [Zeitpunkt der Sicherung](#)
 - [Inkrementelle Sicherung](#)
 - [Sicherheit der Sicherung](#)
 - [Wiederherstellung ausprobieren](#)
-

Vorüberlegungen

Die Datensicherung ist meist mit nicht unerheblichem Aufwand verbunden. Das Sichern der Datenbestände, die durch den billigen Plattenplatz immer umfangreicher werden, erfordert natürlich Zeit. Dem steht die Forderung nach ständiger Verfügbarkeit der EDV entgegen. Da diese Aufgabe so wichtig ist, ist eine durchdachte Planung erforderlich, die auch berücksichtigt, mit welcher Geschwindigkeit ein Weiterarbeiten nach dem Crash möglich sein soll.

Die Unternehmensdaten werden heutzutage meist in Datenbanken abgelegt, die ihre eigene Datensicherung mitbringen. Die Datenbanken erlauben selbst im laufenden Betrieb eine verlustfreie Sicherung der Daten. So konzentriert sich die selbst gestaltete Sicherung auf die Sicherung des Betriebssystems mit seinen Konfigurationsdaten und der anderen Anwenderdaten wie beispielsweise Mailboxen oder Schriftverkehr.

Zeitpunkt der Sicherung

Die Häufigkeit der Sicherung hängt natürlich von dem Schaden ab, der entsteht, wenn ein bestimmter Zeitraum an Arbeit verloren geht. Bei einem Softwarehersteller ist der Verlust von einigen Tagen zwar extrem ärgerlich, aber nicht existenzbedrohend. Meist erinnern sich die Entwickler noch recht gut, was sie alles in der Zeit geändert haben, und können das recht schnell nachtragen. Dagegen ist bei einem Callcenter, das telefonische Bestellungen entgegennimmt, bereits der Verlust einer Stunde dramatisch, wenn es keine zusätzlichen schriftlichen Aufzeichnungen über die Bestellungen gibt.

Da die Datensicherung auch einen Archivierungscharakter hat, muss man wissen, wie lange Sicherungen zurückreichen müssen. Üblich ist es, Wochen-, Monats- und Quartalssicherungen länger aufzuheben.

Inkrementelle Sicherung

Eng verbunden mit dem Zeitpunkt der Sicherung ist die Frage, wie lange die Sicherung dauert. Selbst beim Einsatz schnellster Hardware ist die Menge der Daten nicht immer problemlos in der Zeit unterzubringen, in der die Maschine brachliegt. Dann ist der Gedanke nahe liegend, nicht die gesamte Datenmenge täglich zu sichern, sondern nur den Teil der Daten, der sich seit gestern verändert hat. Dann könnte man beispielsweise am Wochenende eine Komplettsicherung erstellen und jeden Tag die Differenz zum Vortag speichern.

Um eine solche Sicherungsstrategie zu entwickeln, verwendet man verschiedene Sicherungslevel, die

ganzzahlig durchnummeriert sind und bei 0 beginnen. Level 0 ist eine Komplettsicherung aller Dateien. Eine Level-i-Sicherung sichert alle Dateien, die seit der letzten Sicherung mit einem Level, der kleiner ist als i, verändert wurden. Eine simple Strategie wäre, wenn Sie einmal im Jahr eine Level-0-Sicherung durchführen, jeden Monat eine Level-3-Sicherung, jede Woche eine Level-5-Sicherung und an den verbleibenden Tagen Level-7-Sicherung verwenden.

Diese Level werden von dem Programm dump direkt unterstützt, das den Level als ersten Parameter erwartet. Alle anderen Sicherungstools bieten aber ebenfalls die Möglichkeit, Dateien zu sichern, die seit einem gewissen Zeitpunkt verändert wurden. Damit kann man jedenfalls vergleichbare Strategien verwenden, muss allerdings genauer verfolgen, welche Datensicherung zu welchem Zeitpunkt erfolgte.

Sicherheit der Sicherung

Wer kontrolliert die Sicherung? Woran merkt man, dass es einen Bandfehler gegeben hat oder dass das Sicherungsprogramm abgestürzt ist? Diese Fragen sind besonders interessant, wenn die Datensicherung unbeobachtet abläuft. Wenn eine Datensicherung einen Bandfehler oder Ähnliches hatte, findet sich ein Hinweis in der Protokolldatei des syslog-Dämons (siehe S. syslog). Um sicherzugehen, dass im vollen Umfang gesichert wurde, leitet man die Aufzählung der gesicherten Dateien in eine eigene Protokolldatei um. Mit Hilfe des Wortzählers wc (siehe S. wc) kann man ermitteln, ob die Anzahl der Dateien den Erwartungen entspricht. Wird der Befehl date zu Anfang und am Ende der Datensicherung abgesetzt, kann erstens überprüft werden, ob die Datensicherung gestartet wurde und nicht abstürzte, und zweitens kann man anhand der Dauer auch feststellen, ob nicht plötzlich erhebliche Differenzen auftreten.

Wenn man die Sicherungen im Regal direkt neben dem Computer aufbewahrt, wird man nach einem Brand oder Diebstahl über keine Daten mehr verfügen. Die simpelste Sicherung dagegen ist, dass ein Angestellter das letzte Exemplar der Datensicherung einfach mit nach Hause nimmt.

Nach einem vollständigen Verlust des Computers durch Brand oder Diebstahl ist eine entscheidende Frage, ob man schnell die Laufwerke, mit denen die Datensicherung erstellt worden ist, auch wieder beschaffen kann. So sind beispielsweise durchaus nicht alle DAT-Streamer miteinander kompatibel, obwohl sie die gleichen Bänder verwenden. Wenn man eine spezielle Software zum Sichern verwendet, sollte ein Duplikat davon sicher verwahrt sein, sonst kann man seine Bänder vielleicht nicht mehr aufspielen.

Wiederherstellung ausprobieren

Es ist gar keine schlechte Idee, den Ernstfall am Wochenende einmal durchzuspielen. Man beschafft am besten ein paar leere Platten und versucht, so schnell wie möglich den letzten Zustand wieder herzustellen. Auch die Rücksicherung einzelner Dateien sollte man ein paarmal ausprobiert haben. Wenn erst einmal ein Ernstfall eintritt, liegt eine Spannung in der Luft, die wenig Freiraum zum Nachdenken lässt. Hier ist dann Routine gefragt.

Unterabschnitte

- [Steuerung eines Bandlaufwerkes: mt](#)

Das Bandlaufwerk

Leider sind die Einträge für das Bandlaufwerk im Verzeichnis /dev nicht auf allen Systemen gleich, und so ist es nicht ganz leicht, das Bandlaufwerk zu erkennen. Spätestens hier kommt man nicht umhin, in das Systemhandbuch zu schauen. Für jeden Bandtyp ist normalerweise ein eigenes Device eingerichtet. Hinzu kommt, dass für das nicht rückspulende Laufwerk ein weiterer Eintrag, üblicherweise mit einem n davor oder hinter den Namen vorgenommen wird. Tabelle zeigt zwei Beispiele für Namen von SCSI-Bandlaufwerken.

[Namen von Bandgeräten]L|L|L Band rückspulend & Band ohne Spulen & System
/dev/rmt/1 & /dev/rmt/1n & Sun Solaris
/dev/rmt/0mb & /dev/rmt/0mnb & HP-UX

Die Geschwindigkeit eines Bandlaufwerk ist durchaus relevant. Die Datensicherung ist meist nicht in wenigen Minuten getan und setzt die Maschine während dieser Zeit weitgehend außer Gefecht. Darum sollten sowohl das Laufwerk als auch der Controller und der Computer höchste Leistung bringen.

Steuerung eines Bandlaufwerkes: mt

Zur Ansteuerung des Bandlaufwerks wird das Programm mt verwendet. Unter MacOS X ist mt nicht verfügbar. (magnetic tape) verwendet. Als Bandlaufwerk verwendet mt den Inhalt der Umgebungsvariablen TAPE oder das Device /dev/tape, das üblicherweise ein symbolischer Link auf das Standardlaufwerk ist. Man kann das Laufwerk auch mit der Option -f direkt angeben:

```
mt -f /dev/gpkursivBandlaufwerk Befehl
```

Der Befehl `mt status` liefert Hinweise, ob das Laufwerk überhaupt ansprechbar ist. Der `mt-Befehl retention` spult das Band einmal nach vorn und wieder zurück. Soll nur zurückgespult werden, gibt es den Befehl `rewind`.

[Kommandos des mt-Befehls]L|L mt-Kommando & Wirkung
`status` & testet, ob das Bandlaufwerk ansprechbar ist
`rewind` & spult das Band zurück
`retention` & spult das Band komplett nach vorn und wieder zurück
`erase` & löscht das einliegende Band

Unterabschnitte

- [Daten zurück: restore](#)
- [Beispiel für eine Fernsicherung mit dump](#)

dump

dump ist ein recht altes Werkzeug in der UNIX-Umgebung. Es ist speziell auf Datensicherung ausgelegt. Vor allem ist es in der Lage, komplette Dateisysteme zu sichern und inkrementelle Datensicherungsstrategien zu unterstützen.

Unter Solaris heißt der Befehl dump ufsdump. Der dort existierende Befehl dump hat eine andere Aufgabe.

Nach dem Befehl dump wird mit einer Ziffer der Level der Sicherung bestimmt. Es folgen weitere Optionen.

Wird die Option u verwendet, wird in der Datei dumpdates im Verzeichnis /var/adm oder /etc hinterlegt, wann welches Dateisystem gesichert worden ist.

Die Option f erwartet einen weiteren Parameter, der den Ort des Bandes beschreibt. Hier kann ein anderes als das Standardband angegeben werden. Es kann durch die Notation host:/dev/tape auch eine Sicherung über das Netz auf dem Computer host durchgeführt werden.

Der letzte Parameter gibt das zu sichernde Dateisystem an. In einigen Versionen von dump funktioniert alternativ die Angabe des mount points.

Mit dem folgenden Sicherungslauf wird das Dateisystem /dev/vg00/lvol3 auf das Standardband gesichert:

```
hpsrv# dump 0 /dev/vg00/lvol3
/var/adm/dumpdates: No such file or directory
DUMP: Date of this level 0 dump: Sun Dec 30 15:27:44 2001
DUMP: Date of last level 0 dump: the epoch
DUMP: Dumping /dev/vg00/rlvol3 (/home) to /dev/rmt/0m
DUMP: This is an HP long file name filesystem
DUMP: mapping (Pass I) [regular files]
DUMP: mapping (Pass II) [directories]
DUMP: estimated 1001 tape blocks on 0.03 tape(s).
DUMP: dumping (Pass III) [directories]
DUMP: dumping (Pass IV) [regular files]
DUMP: DUMP: 1001 tape blocks on 1 tape(s)
```

```
DUMP: DUMP: /dev/vg00/rlvol3 has 0 inodes with unsaved ...
DUMP: DUMP IS DONE
DUMP: Tape rewinding
hpsrv#
```

Sollen mehrere Dateisysteme auf ein Band geschrieben werden, muss das Device des Bandgerätes so gewählt werden, dass es nicht zurückspult.

Daten zurück: restore

Der Befehl `restore` holt gesicherte Datenbestände zurück.

Die Option `-r` sorgt für das komplette Einspielen der gesicherten Daten. Zuvor wird das Dateisystem, auf das die Daten sollen, neu mit `mkfs` (siehe S. `mkfs`) erzeugt und per `mount` eingehängt. Mit `cd` wechselt man in das Verzeichnis, unter dem das Dateisystem eingehängt wurde, und ruft `restore` mit der Option `-r` auf. Bei einer inkrementellen Datensicherung sind auch die neuesten Bänder der jeweiligen Levels einzuspielen.

Mit der Option `-i` können interaktiv einzelne Dateien zurückgeholt werden. `restore` holt den Katalog vom Band und setzt den Benutzer auf einen eigenen Prompt, in dem er die Standardbefehle `ls`, `pwd` und `cd` verwenden kann, als sei er auf einem Dateisystem. Mit dem Befehl `add Datei` wird die genannte Datei in die Liste der Dateien aufgenommen, die wieder zurückgeholt werden sollen. Mit dem Befehl `extract` startet das Zurückholen der Daten.

Im Beispiel wird die Datei `hello.cpp` aus dem Heimatverzeichnis zurückgeholt. Zunächst wechselt man in das Verzeichnis, wo die gesicherten Dateien später abgelegt werden sollen. In diesem Fall wurde ein Verzeichnis `restore` unter `/var` angelegt:

```
hpsrv# cd /var
hpsrv# mkdir restore
hpsrv# cd restore
hpsrv# restore i
restore > cd arnold
restore > ls
./arnold:
.cshrc      .history   .profile   .sh_history bad.tif
.exrc       .login     .rhosts    a.out      hello.cpp
```

```
restore > add hello.cpp
restore > ls
./arnold:
.cshrc      .history   .profile   .sh_history bad.tif
.exrc       .login     .rhosts    a.out      *hello.cpp
```

```
restore > extract
You have not read any tapes yet.
Unless you know which volume your file(s) are on you should
start with the last volume and work towards the first.
Specify next volume #:
Specify next volume #: 1
set owner/mode for '.'? [yn] n
restore > quit
hpsrv# ls
arnold
hpsrv# pwd
/var/restore
hpsrv# ll arnold
```

```
total 2
-rw-rw-rw-  1 arnold  users      45 Dec 20 21:43 hello.cpp
hpsrv#
```

Wie Sie sehen, werden die Dateien in einem Abbild der gesicherten Strukturen auf dem aktuellen Pfad abgelegt. Die Datei hello.cpp muss also noch in das Originalverzeichnis umkopiert werden.

Beispiel für eine Fernsicherung mit dump

Mit dump kann man den eigenen Datenbestand auch über das Netz auf dem Bandgerät einer anderen Maschine sichern. Der rsh-Dämon (siehe S. rshd) muss auf der Zielmaschine verfügbar und korrekt installiert sein. Bei einer Fernsicherung wird das Programm rmt auf dem entfernten Rechner benötigt, das von dump in /etc/dump gesucht wird. Auf der HP-UX-Maschine, die in diesem Fall als Zielmaschine dienen soll, befindet sich rmt aber im Verzeichnis /usr/sbin. Durch die Environmentvariable RMT kann der Pfad allerdings auf der Quellmaschine korrigiert werden:

```
gaston# RMT=/usr/sbin/rmt
gaston# export RMT
gaston# dump -0 -f hpsrv:/dev/rmt/0mb /dev/hda7
DUMP: Connection to hpsrv established.
DUMP: Date of this level 0 dump: Mon Feb  4 12:14:18 2002
DUMP: Dumping /dev/hda7 (/home) to /dev/rmt/0mb on host hpsrv
DUMP: Added inode 7 to exclude list (resize inode)
DUMP: Label: none
DUMP: mapping (Pass I) [regular files]
DUMP: mapping (Pass II) [directories]
DUMP: estimated 2598351 tape blocks.
DUMP: Volume 1 started with block 1 at: Mon Feb  4 12:14:20 2002
DUMP: dumping (Pass III) [directories]
DUMP: dumping (Pass IV) [regular files]
DUMP: 1.65% done at 142 kB/s, finished in 4:58
...
DUMP: 75.61% done at 176 kB/s, finished in 0:59
DUMP: write: No space left on device
DUMP: End of tape detected
DUMP: write: No space left on device
DUMP: write: No space left on device
DUMP: Closing /dev/rmt/0mb
DUMP: Volume 1 completed at: Mon Feb  4 15:21:28 2002
DUMP: volume 1 1970350 tape blocks (1924.17MB)
DUMP: volume 1 took 3:07:08
DUMP: volume 1 transfer rate: 175 kB/s
DUMP: Change Volumes: Mount volume #2
DUMP: Is the new volume mounted and ready to go?: ("yes" or "no") yes
DUMP: volume 2 started with block 1970321 at: Mon Feb  4 16:57:21 2002
DUMP: volume 2 begins with blocks from inode 611797
DUMP: 75.83% done at 39 kB/s, finished in 0:59
...
DUMP: 98.40% done at 177 kB/s, finished in 0:03
DUMP: 100.00% done at 177 kB/s, finished in 0:00
DUMP: Closing /dev/rmt/0mb
DUMP: volume 2 completed at: Mon Feb  4 18:00:53 2002
DUMP: volume 2 668010 tape blocks (652.35MB)
DUMP: volume 2 took 1:03:32
DUMP: volume 2 transfer rate: 175 kB/s
DUMP: 2638330 tape blocks (2576.49MB) on 2 volume(s)
DUMP: finished in 14988 seconds, throughput 176 kBytes/sec
DUMP: Date of this level 0 dump: Mon Feb  4 12:14:18 2002
DUMP: Date this dump completed: Mon Feb  4 18:00:53 2002
DUMP: Average transfer rate: 175 kB/s
```

```
DUMP: DUMP IS DONE
gaston#
```

Interessant an diesem dump ist, dass das Band zu Ende war. Daraufhin wird ein neues Medium angefordert und auf diesem weiter geschrieben. dump fordert also automatisch zum Medienwechsel auf, wenn das Band voll ist.

Die Stärken und Schwächen von dump liegen darin, dass es alle Dateien eines Dateisystems sichert. Man ist also nicht frei in der Wahl der Verzeichnisse, die man sichern will. Soll also beispielsweise neben den im Beispiel verwendeten Heimatverzeichnissen noch das Spoolverzeichnis des Mailsystems gesichert werden, ist dump überfordert. Es sei denn, das Spoolverzeichnis liegt ebenfalls auf einem eigenen Dateisystem.

Die Fähigkeit von dump, Sicherungslevel zu verwalten, macht inkrementelle Sicherungen natürlich wesentlich einfacher. Sehr elegant ist die interaktive Rücksicherung einzelner Dateien.

« [Das Bandlaufwerk](#) | [Datensicherung](#) | [tar \(tape archiver\)](#) »

Unterabschnitte

- [Zusammensetzung des tar-Kommandos](#)
 - [Die Steuerungsdatei /etc/default/tar](#)
 - [Netzwerksicherung per tar](#)
-

tar (tape archiver)

Trotz seines Namens kann tar nicht nur mit dem Band, sondern auch mit Disketten, Wechselmedien und sogar Dateien als Ziel arbeiten. Der Vorteil von tar ist seine weite Verbreitung und Verfügbarkeit. tar wurde im Laufe der Jahre weiterentwickelt. Die Vorteile von tar sind:

- tar ist auf jeder Maschine verfügbar. Es gibt sogar einige Implementierungen auf anderen Betriebssystemen.
- tar ist eine ausgetestete Software.
- tar kann eine Sicherung über mehrere Medien verteilen
- tar ist in vielen Implementierungen netzwerkfähig.

Die Nachteile von tar sind:

- tar kann (in seiner Grundversion) keine Gerätedateien und andere spezielle Dateien sichern, ist also nicht für die Systemsicherung geeignet.
- tar ist unflexibel bei der Rücksicherung einzelner Dateien.

Zusammensetzung des tar-Kommandos

Nach dem Befehl tar bestimmt der erste Buchstabe, welche Operation ausgeführt wird. Hier muss einer der Buchstaben c, x oder t auftauchen. Dabei bedeutet:

[Operationsoptionen von tar]C|L Zeichen & Bedeutung

c & Erzeuge ein Archiv

x & Entpacke ein Archiv

t & Lies das Inhaltsverzeichnis eines Archivs

Bei den eigentlichen Optionen sind die folgenden von Bedeutung:

- [f sicherungsdatei] Mit f wird angegeben, dass die Sicherung in eine Datei erfolgen soll. Als Dateiname wird dann die Sicherungsdatei oder das Device angegeben, auf das gesichert werden soll.
- [v] zeigt alle gesicherten Dateien an. Daraus lässt sich leicht ein Sicherungsprotokoll erstellen.
- [z oder Z] gibt an, dass das Archiv komprimiert wird. Dadurch passt natürlich mehr auf das Band. Bei Maschinen mit schwacher CPU-Leistung ist allerdings zu prüfen, ob der Aufwand für die Komprimierung das Schreiben so weit verzögert, dass der Datenstrom abreißt. Das hat beim direkten Schreiben auf Bänder den Effekt, dass die Datensicherung um ein Vielfaches länger dauert. Bandgeräte brauchen einen möglichst kontinuierlichen Dateninput. Sobald eine Unterbrechung stattfindet, muss das Band stoppen, kurz zurückfahren, neu positionieren, um dann wieder zu starten, wenn neue Daten eintreffen.

- [M] arbeitet mit Medienwechsel. Ist beim Sichern das Medium voll, wird der Benutzer aufgefordert, das Medium zu wechseln und die Return-Taste zu drücken. Beim Rücksichern werden die erforderlichen Medien automatisch angefordert.

Als Beispiel soll hier das Verzeichnis /home gesichert werden. Ohne weitere Angabe verwendet tar das Standardbandgerät:

```
# cd /home
# tar cv .
a ./tacoss symbolic link to /users/tacoss
a ./willemer symbolic link to /users/willemer
a ./notes/.sh_history 4 blocks
a ./notes/.profile 1 blocks
a ./notes/server symbolic link to /opt/lotus/bin/server
a ./notes/.Xpdefaults 2 blocks
a ./notes/ console.tmp 1 blocks
a ./rossow/.sh_history 1 blocks
a ./rossow/.rhosts 1 blocks
a ./arnold/.cshrc 2 blocks
a ./arnold/.exrc 1 blocks
a ./arnold/.login 1 blocks
a ./arnold/.profile 1 blocks
a ./arnold/.sh_history 5 blocks
a ./arnold/.rhosts 1 blocks
a ./arnold/bad.tif 1813 blocks
a ./arnold/a.out 41 blocks
a ./arnold/hello.cpp 1 blocks
a ./arnold/.history 1 blocks
#
```

Die vorangegangene Ausgabe von tar zeigt, dass diese Version von tar relative Pfade abspeichert. Manche Versionen speichern immer den absoluten Pfad. Sollen die Dateien beim Restaurieren an eine andere Stelle kommen, empfiehlt es sich, die Option A zu probieren. Sie entfernt bei solchen Systemen normalerweise den führenden Schrägstrich beim Sichern und Zurücksichern.

Das Zurückholen aller Dateien ist relativ einfach. Man wechselt per cd in das Zielverzeichnis und ruft tar mit der Option x auf. Wer sehen möchte, was zurückgeholt wird, gibt noch zusätzlich ein v an.

Sollen dagegen bestimmte Dateien zurückgeholt werden, ist das etwas komplizierter. Hinter tar xv kann man die gewünschten Dateien angeben, die man zurückholen möchte. Allerdings wünscht sich tar den kompletten Pfad, so wie er beim Sichern angegeben wurde. Will man also die Datei arnold/hello.cpp zurückholen, lautet der Befehl:

```
# tar xv ./arnold/hello.cpp
x ./arnold/hello.cpp, 64 bytes, 1 tape blocks
#
```

Dagegen scheitert der Versuch, als Parameter arnold/hello.cpp anzugeben, obwohl das inhaltlich das Gleiche ist. Unter diesem Namen ist es eben nicht gespeichert worden. Sämtliche Versuche, diese Version von tar dazu zu bewegen, mit Wildcards zu arbeiten scheitern. Das einzige Zugeständnis ist, dass man den Pfadnamen ./arnold verwenden kann und alle darunter liegenden Dateien und Verzeichnisse restauriert werden:

```
# tar xv ./arnold
x ./arnold/.cshrc, 814 bytes, 2 tape blocks
```

```
x ./arnold/.exrc, 347 bytes, 1 tape blocks
x ./arnold/.login, 341 bytes, 1 tape blocks
x ./arnold/.profile, 446 bytes, 1 tape blocks
x ./arnold/.sh_history, 2538 bytes, 5 tape blocks
x ./arnold/.rhosts, 7 bytes, 1 tape blocks
x ./arnold/bad.tif, 928109 bytes, 1813 tape blocks
x ./arnold/a.out, 20524 bytes, 41 tape blocks
x ./arnold/hello.cpp, 64 bytes, 1 tape blocks
x ./arnold/.history, 212 bytes, 1 tape blocks
#
```

Man ahnt es schon: Der nackte Parameter `arnold` ohne `./` scheitert. Das besonders Ärgerliche an solchen scheiternden Versuchen ist, dass man sie erst feststellt, wenn das ganze Band durchsucht wurde, was einige Zeit dauern kann.

Einige Versionen von `tar` können mehr. Das betrifft insbesondere die GNU-Version. Hier ein Beispiel:

```
gaston> tar xvf /dev/fd0 "*syslog.tex"
unprog/syslog.tex
gaston>
```

GNU `tar` hat keinerlei Probleme mit Wildcards. Wer oft mit `tar` arbeitet und häufiger einzelne Dateien zurückholen muss, sollte prüfen, ob es ein GNU `tar` für sein System gibt.

Fazit: Gerade das Zurückholen einzelner Dateien ist nicht trivial und von der Implementierung des `tar` abhängig. Hier sollte man sich vor dem Fall der Fälle aus den Manpages und dem Systemhandbuch informieren und am besten den Vorgang einmal ausprobieren.

Die Steuerungsdatei `/etc/default/tar`

Die `tar`-Implementation von SCO und SINIX verwendet eine Steuerungsdatei namens `/etc/default/tar`. Dort sind die verschiedenen Diskettenvarianten und einige Bandgeräte mit Kapazität und Device durchnummeriert aufgezählt. Die entsprechende Zahl wird dann im `tar`-Aufruf verwendet. Der Vorteil dieses Verfahrens ist, dass `tar` die Größe seiner Bänder kennt.

Nehmen wir als Beispiel an, es soll das Verzeichnis `/usr/bin` mit allen Unterverzeichnissen gesichert und wieder zurückgeladen werden. Zunächst wird mit dem Befehl `cd /` in das Rootverzeichnis gewechselt. Dann wird der jeweils passende Befehl aus Tabelle verwendet.

[Variationen im `tar`]
Linux & `tar cf /dev/tape /usr/bin & tar xf /dev/tape`
SCO & `tar c8 /usr/bin & tar x8`

Netzwerksicherung per `tar`

Nicht immer dokumentiert, aber meist implementiert ist, dass `tar` in der Lage ist, Dateien über das Netzwerk zu sichern. Dabei wird `tar` als Bandlaufwerk das Device einer anderen Maschine als Ziel der Datensicherung angegeben. Dabei wird der Hostname, gefolgt von einem Doppelpunkt und dem dortigen Device, als Ziel verwendet. Diese Namensgebung für Netzobjekte folgt der des Befehls `rcp` (remote copy; siehe S. `rcp`). Die Nomenklatur kommt nicht von ungefähr, da beide über eine Pipe über den `rshd` (siehe S. `rshd`) implementiert sind.

```
tar cf idfix:/dev/tape /usr/bin
```

Das lokale Verzeichnis /usr/bin wird auf das Bandlaufwerk der Maschine idfix gesichert. Da die Sicherung über den rshd erfolgt, muss natürlich auch die Berechtigungskonfiguration in der Datei .rhosts stimmen.

« [dump](#) | [Datensicherung](#) | [cpio](#) »

Unterabschnitte

- [Sichern, anschauen und zurückholen](#)
- [Wichtige Optionen](#)
- [Verzeichniskopie via cpio](#)

cpio

`cpio` ist ein weiteres Programm zur Datensicherung. Es ist sehr flexibel, was die Bestimmung der Dateien angeht, die zu sichern sind, und der Dateien, die wieder zurückzuholen sind. `cpio` hat eine etwas ungewohnte Syntax, da es die zu sichernden Dateien dem Standardeingabekanal entnimmt.

`cpio` behält in der Standardversion weder Uhrzeit noch Eigner oder Rechte der Dateien. Per Option kann allerdings für die gesicherte Datei der Originalzeitpunkt nach der Rücksicherung wiederhergestellt werden. Die Gerätedateien und andere spezielle Dateien lassen sich nur mit Administratorrechten und entsprechend gesetzter Option sichern bzw. wiederherstellen.

Sichern, anschauen und zurückholen

`cpio` entnimmt die Namen der zu sichernden Dateien der Standardeingabe. Dabei muss jeder Dateiname in einer eigenen Zeile stehen. Damit steht der Befehl `cpio` typischerweise an zweiter Stelle in einer Pipe und hat beim Sichern die Option `-o`. An erster Stelle kann jeder beliebiger UNIX-Befehl stehen, der eine Dateiliste erzeugt. Das kann `ls` sein, dann werden die Dateien des aktuellen Verzeichnisses gesichert, allerdings nicht dessen Unterverzeichnisse. Will man alle Dateien im Unterverzeichnis sichern, stellt man den Befehl `find` voran. Durch dessen flexible Möglichkeiten kann sehr fein selektiert werden, welche Dateien gesichert werden sollen. Bei ganz besonders kuriosen Dateizusammenstellungen kann man die Dateiliste auch in eine Datei stellen und dessen Inhalt mit dem Befehl `cat` ausgeben.

Das Ergebnis, also die Ausgabe des `cpio`, geht auf `stdout` und wird durch die Ausgabeumlenkung (`>`) auf das Sicherungsdevice oder die Archivdatei umgeleitet. Wieder soll das Verzeichnis `/home` mit allen Unterverzeichnissen gesichert werden:

```
hpsrv# cd /home
hpsrv# find . -print | cpio -o >/dev/rmt/0mb
(Using tape drive with immediate report mode enabled (reel #1).)
1870 blocks
hpsrv#
```

Dieser Befehl sichert alle Dateien unterhalb des aktuellen Verzeichnisses auf das Bandgerät. Nun soll betrachtet werden, was alles auf dem Band vorhanden ist. Dazu wird die Option `-it` verwendet. Das Sicherungsgerät wird als Standardeingabe verwendet:

```
hpsrv# cpio -it </dev/rmt/0mb
.
```

```

lost+found
tacoss
willemer
notes
notes/.sh_history
notes/.profile
notes/server
notes/.Xpdefaults
notes/ console.tmp
rossow
rossow/.sh_history
rossow/.rhosts
arnold
arnold/.cshrc
arnold/.exrc
arnold/.login
arnold/.profile
arnold/.sh_history
arnold/.rhosts
arnold/bad.tif
arnold/a.out
arnold/hello.cpp
arnold/.history
1870 blocks
hpsrv#

```

Um die ganze Sicherung zurückzuholen, wird fast die gleiche Befehlskombination verwendet, wie beim Auslesen des Verzeichnisses. `-i` bewirkt das Lesen der Sicherung, und `-d` bewirkt, dass auch Verzeichnisse erstellt werden. Das Auslesen geschieht im aktuellen Verzeichnis. Vor dem Auspacken muss also in das Zielverzeichnis gewechselt werden.

```

hpsrv# cpio -id </dev/rmt/0mb
1870 blocks
hpsrv#

```

Im letzten Beispiel soll nun als einzige Datei `hello.cpp` vom Band geholt werden. An dieser Stelle zeigt `cpio` seine große Flexibilität. Im Anschluss an die Option wird in Hochkommata das Muster der gesuchten Dateinamen angegeben. Dabei können Wildcards verwendet werden.

```

hpsrv# rm -r *
hpsrv# cpio -id '*hello*' </dev/rmt/0mb
1870 blocks
hpsrv# ll
total 2
drwxrwxrwx  2 arnold  users      1024 Dec 31 01:46 arnold
hpsrv# ll arnold
total 2
-rw-rw-rw-  1 arnold  users       64 Dec 31 01:46 hello.cpp
hpsrv#

```

Wichtige Optionen

Je nach Version des `cpio` gibt es unterschiedliche Optionen. Genauerer findet sich in den Manpages. Die hier genannten Optionen dürften in jeder Version zu finden sein:

- `[m]` Behält die Modifikationszeit des gesicherten Originals bei. Wirkt nicht auf Verzeichnisse.

- [d] Erzeugt beim Rücksichern die notwendigen Verzeichnisse.
- [u] Ersetzt durch die Rücksicherung auch eine neuere Datei als die, die im Archiv steht.
- [x] Sichert auch Gerätedateien und andere spezielle Dateien.

Verzeichniskopie via cpio

Auch mit dem Befehl `cpio` lassen sich komplette Verzeichnisse kopieren. Dabei ist zu beachten, dass alte Versionen von `cpio` bei der Kopie die Dateieigenschaften nicht erhalten können:

```
find Quellverzeichnis -depth -print | cpio -pdm Zielverzeichnis
```

In der Praxis wird man in das Quellverzeichnis wechseln und als Parameter einen Punkt angeben. Ansonsten wird der Quellverzeichnisname mit übertragen und an das Zielverzeichnis angehängt.

« [tar \(tape archiver\)](#) | [Datensicherung](#) | [Medien kopieren: dd](#) »

Medien kopieren: dd

dd ist ein Kopierprogramm, das von »Input File« (if) nach »Output File« (of) schreibt. Dabei kann der In- und Output sowohl eine Datei als auch ein Device sein. Auf diese Weise kann dd eine Diskette auf eine Datei übertragen, um sie hinterher wieder auf eine Diskette zu transferieren.

```
dd if=/dev/fd0 of=dumpfile  
dd of=/dev/fd0 if=dumpfile
```

dd muss das Format, mit dem das Medium beschrieben wurde, nicht unbedingt beherrschen, es muss die Diskette nur direkt lesen können. So kann man sogar Mac-Disketten auf einer Linux-Maschine kopieren. Da die von dd erzeugte Datei (hier dumpfile) eine Datei ist wie jede andere, kann man sie per E-Mail verschicken oder auf CD brennen und kann jederzeit wieder eine Diskette daraus machen. Dies ist besonders bei bootfähigen Disketten interessant.

[« cpio](#) | [Datensicherung](#) | [Andere Sicherungstools: AMANDA](#) »

Andere Sicherungstools: AMANDA

Das Open-Source-Projekt AMANDA (Advanced Maryland Automatic Network Disk Archiver) ist ein Werkzeug zum Sichern kompletter Netzwerkumgebungen. Im Zentrum einer solchen Architektur steht der Backupserver, der die Backupmedien hat. Er zieht von den angeschlossenen Clients die zu sichernden Images über das Netzwerk, parkt sie gegebenenfalls auf lokalen Platten zwischen und schreibt sie anschließend auf Band. AMANDA hat kein eigenes Datensicherungsformat, sondern verwendet die maschineneigenen Datensicherungsmethoden.vgl. Nemeth, Evi / Snyder, Garth / Seebass, Scott / Hein, Trent R.: UNIX Systemverwaltung. Markt+Technik - Prentice Hall, München, 2001. S. 257-273 sowie Schmidt, Fabian: Backup ohne Klicks. Linux Magazin 05/2002. S. 52-54.

AMANDA ist als Quelltextdistribution erhältlich und muss selbst übersetzt werden. Nach dem Auspacken wird mit dem beigefügten `configure` die Software an die Maschine angepasst. Dabei müssen auch der Benutzer und die Gruppe angegeben werden. Soll dies unter dem Benutzer `amanda` und der Gruppe `dasi` erfolgen, wird folgender Aufruf abgesetzt:

```
hpsrv# ./configure -with-user=amanda -with-group=dasi
```

Anschließend werden die Quellen übersetzt und die erzeugten Binaries installiert. Dazu braucht die Zielmaschine mindestens einen C-Compiler sowie `lex` und `yacc` bzw. deren GNU-Varianten `flex` und `bison`.

```
hpsrv# make
hpsrv# make install
```

Standardmäßig sucht AMANDA ihre Konfigurationsdatei `amanda.conf` im Verzeichnis `/var/lib/amanda/DailySet1`. S.u.S.E legt die Konfiguration in das Verzeichnis `/etc/amanda/DailySet1`. Als gute Vorlage gibt es eine Beispielkonfiguration. Dabei sind einige Einträge anzupassen. In erster Linie muss das Bandlaufwerk eingetragen werden. Hier wird mit dem Parameter »holdingdisk« eingestellt, dass die Platte als Zwischenspeicher verwendet werden soll. Schließlich wird beispielsweise `comp-root-tar` als Art der Datensicherung definiert. In der Definition wird sich wiederum auf `root-tar` bezogen. Eine Vielzahl von Beispielen ist in der Datei schon vorhanden, sodass Sie wahrscheinlich sogar etwas Passendes finden.

```
tapetype HP-DAT
tapedev "/dev/rtm/0mnb" # war: "/dev/null"
```

```
holdingdisk hd1 {
    directory "/u/amanda"
    use 1000 Mb
}
```

```
define dumptype root-tar {
    global
    program "GNUTAR"
    comment "root partitions dumped with tar"
    compress none
    index
```

```
exclude list "/usr/local/lib/amanda/exclude.gtar"
priority low
}
```

```
define dumptype comp-root-tar {
    root-tar
    comment "Root partitions with compression"
    compress client fast
}
```

Die Datei `/var/lib/amanda/.amandahosts` enthält eine Liste der Anwender und ihre Hosts, die Zugriffsrechte haben. Zum Beispiel:

gaston	amanda
gaston	root
hpsrv	amanda
hpsrv	root

Im gleichen Verzeichnis befindet sich eine Datei `disklist`, die alle zu sichernden Platten und die Maschinen angibt. Der erste Eintrag betrifft die Maschine `sol`. Das Plattendevise heißt `/dev/sd0a`. Der Sicherungstyp »comp-root« wird in der Datei `amanda.conf` festgelegt. Die zweite Zeile sichert das Dateisystem `/dev/hda5` nach dem gleichen Verfahren:

sol	sd0a	comp-root-tar
gaston	hda5	comp-root-tar

Im Client-Server-Betrieb müssen natürlich die passenden Einträge in die Datei `/etc/services` geschrieben werden.

amanda	10080/tcp	# AMANDA backup services
amanda	10080/udp	# AMANDA backup services

Und ein Eintrag in die Datei `/etc/inetd.conf` ist auch erforderlich.

amanda	dgram	udp	wait	amanda	/usr/lib/amanda/amandad	amandad
--------	-------	-----	------	--------	-------------------------	---------

Für Rechner mit MS Windows gibt es keinen Client für AMANDA. Man kann sie sichern, indem man sie per SAMBA (siehe S. samba) ins Netz holt und von der entsprechenden UNIX-Maschine aus sichert.

Nach der erfolgreichen Konfiguration wird das ganze Gebilde geprüft. Dazu melden Sie sich als Benutzer `amanda` an und verwenden den Befehl `amcheck`, der als Parameter den reinen Dateinamen ohne Pfad der Konfigurationsdatei haben muss. Im Beispiel ist das `DailySet1`. Mit dem Befehl `amlabel` wird dem Band ein Name gegeben. Mit dem Befehl `amdump` erfolgt die eigentliche Datensicherung. Normalerweise wird man diesen Befehl in der `crontab` unterbringen.

Die Datenrückholung erfolgt mit dem Befehl `amrecover`. Das Verfahren läuft dann so ab, wie beim Befehl `recover` der im Abschnitt zu `dump` bereits beschrieben wurde.

Die Homepage von AMANDA bietet nicht nur Möglichkeiten zum Download der Software, sondern auch eine »FAQ-O-Matic« und eine ausführliche Dokumentation.

« [Medien kopieren: dd](#) | **Datensicherung** | [Beispiel für eine Sicherung](#) »

Beispiel für eine Sicherung auf CD-RW

Um die Daten meiner Workstation auf CD-RW zu sichern, habe ich mir ein kleines Skript geschrieben, das ein Verzeichnis mit meinen Daten sichert.

Die beiden Variablen SAVEDIR und STARTDIR müssen an den jeweiligen Bedarf angepasst werden. SAVEDIR ist das Verzeichnis, das alle sichernswerten Daten enthält. Ich habe zu diesem Zweck ein Verzeichnis namens my im Heimatverzeichnis angelegt. Soll eine neue CD-RW bearbeitet werden, wird das Skript mit dem Parameter »blank« aufgerufen. Dann wird die CD-RW gelöscht. Ansonsten gibt es keine Abhängigkeiten vom Medium. Es können also auch einfache CD-Rohlinge benutzt werden.

Da ich nicht so viele Daten zu sichern habe, um eine komplette CD zu füllen, schreibt das Skript mehrere Sicherungen auf eine CD und verwendet dafür Multisession. Um die verschiedenen Sicherungsstände leichter erkennen zu können, wird ein Link angelegt, dessen Name aus dem aktuellen Datum generiert wird. Der Link zeigt auf das Verzeichnis, das die Variable SAVEDIR bezeichnet. Um den Link mitsichern zu können, braucht man ein leeres Verzeichnis, in das der Link gelegt wird und das komplett gesichert werden kann. Dies wird in STARTDIR abgelegt. Vorsicht! Das Skript löscht den Inhalt beim Start, um Reste alter Sicherungen zu beseitigen.

```
#!/bin/sh
# cddasi: Datensicherung auf CD-R und CD-RW
# (C) Arnold willemer 5.1.2002
#
# Das Verzeichnis STARTDIR ist das Arbeitsverzeichnis des
# Skripts.
# !!!!!!! $STARTDIR wird zu Anfang vollständig gelöscht !!!!!!!
# Hier wird ein Link auf das Verzeichnis SAVEDIR gelegt und mit
# dem aktuellen Datum versehen. Auf diese Weise wird bei jeder
# Session ein neues Verzeichnis erzeugt, dessen Datum im
# Verzeichnisnamen steht.
```

```
STARTDIR=/home/arnold/bin/savedir
SAVEDIR=/home/arnold/my
SPEED=2
DEV=0,3,0
```

```
# Räume das Startverzeichnis, und lege den Arbeitslink an
cd
$STARTDIR
rm *
ln -s$
[tex2html_wrap_inline5490]SAVEDIR `date +`
```

```
# Auswertung des ersten Parameters
case "
$1" in
  "blank" )
    mkisofs -J -R -f -o image.iso$
STARTDIR
    cdrecord -v speed=DEV -blank=fast -multi
```

```

        image.iso
;;
"new" )
    mkisofs -J -R -f -o image.iso
$STARTDIR
    cdrecord -v speed=$
SPEED dev=
$DEV -multi image.iso
;;
"last" )
    TRACKPOS=`cdrecord -msinfo dev=0,3,0`
    mkisofs -J -R -f -o image.iso -C$
TRACKPOS -M [tex2html_wrap_inline5500]STARTDIR
    cdrecord -v speed=[tex2html_wrap_inline5502]DEV image.iso
;;
* )
    TRACKPOS=`cdrecord -msinfo dev=0,3,0`
    mkisofs -J -R -f -o image.iso -C [tex2html_wrap_inline5504]DEV
$STARTDIR
    cdrecord -v speed=$
[tex2html_wrap_inline5506]SPEED dev=
$DEV -multi image.iso
esac$
[tex2html_wrap_inline5508]

```

Zu guter Letzt muss natürlich noch vor dem Start das Device für den Brenner, der bei mir die SCSI-ID 3 hat, in der Variablen DEV angepasst werden. In der Variablen SPEED wird festgelegt, wie schnell gebrannt wird.

Das Skript `cddas i` akzeptiert alternativ einen Parameter:

- [blank]
Wird eine CD-RW eingelegt, die gelöscht werden soll, muss dieser Parameter gewählt werden.
 - [new]
Eine neue CD-R oder eine neue CD-RW, die noch nicht beschrieben war, wurde eingelegt. In beiden Fällen wird nicht versucht, die TRACKPOS zu bestimmen, was zu einem Abbruch führen würde.
 - [last]
Egal ob CD-R oder CD-RW, mit dem letzten Sichern sollte die CD abgeschlossen werden. Nach der Verwendung von `last` kann nicht mehr darauf gebrannt werden.
 - [*]
Wird kein Parameter angegeben oder etwas anderes als die oben genannten Parameter, erfolgt eine normale Sicherung. Von der CD wird ermittelt, an welche Position gebrannt werden muss und es wird `per multi` auf die CD-R oder CD-RW gebrannt.
-

« [Andere Sicherungstools: AMANDA](#) | [Datensicherung](#) | [Software installieren](#) »

Software installieren

Die Installation von Software, auch von Anwendersoftware, ist unter UNIX Aufgabe des Administrators. Die Programme und ihre Dateien werden in Bereichen abgelegt, die ein Anwender nicht verändern darf. Dazu gehören beispielsweise /etc oder /usr/bin. Das scheint zunächst umständlich, hat aber den Vorteil, dass kein Anwender und damit auch kein Schädling, den ein Anwender einschleppt, die Programme sabotieren kann.

Unter UNIX haben sich im Laufe der Zeit unterschiedliche Verfahren herausgebildet, um Software zu installieren. Dabei sind folgende Verfahren von Bedeutung:

- [make] Hier wird meist eine gepackte tar-Datei ausgeliefert. Nach dem Auspacken wird das Programm make verwendet, um die Software zu installieren. Der Vorteil dieses Verfahrens ist die Portabilität.
- [Solaris Packages] Die Software ist zu speziellen Paketen gepackt. Sie wird mit dem Befehl pkgadd installiert und mit pkgrm deinstalliert.
- [HP-UX] Der Software-Agent von HP-UX kann zu einem System für eine Ferninstallation ganzer HP-UX-Netze ausgebaut werden.
- [rpm] Diese Software ist unter Linux, insbesondere von Red Hat, entwickelt worden und hat sich weitgehend als Standard in diesem Bereich etabliert. Der zentrale Befehl zur Installation und Deinstallation heißt rpm.

Unterabschnitte

- [make als Installationswerkzeug](#)
- [Solaris Packages](#)
- [HP-UX: SD-UX](#)
- [Red Hat Package Manager](#)
 - [Informationen holen](#)

[« Beispiel für eine Sicherung](#) | [Administration](#) | [make als Installationswerkzeug](#) »

make als Installationswerkzeug

Das Programm `make` (siehe S. `make`) eignet sich für die Installation von Softwarepaketen. Es hat sich inzwischen bewährt, als Parameter `install` anzugeben, um eine Installation durchzuführen. Da inzwischen fast alle Systeme spezielle Programme für die Softwareinstallation besitzen, findet man `make` fast nur noch dort, wo Pakete im Quelltext ausgeliefert werden, also insbesondere im Open-Source-Bereich.

Die Auslieferung im Quelltext bietet den Vorteil, dass man ein Softwarepaket für alle Systeme erstellen kann. Die Software wird normalerweise mit `tar` ausgepackt. Dabei wird gleich ein Packer, meist `gzip` eingebunden. Auf manchen Maschinen funktioniert das nicht, dann muss man zunächst entpacken und dann das Paket mit `tar` zerlegen. Dabei wird normalerweise ein lokales Verzeichnis angelegt, in das man nun wechselt.

Der Autor des Pakets legt oftmals eine README-Datei bei, aus der die weiteren Schritte hervorgehen. Auch wenn das in manchen Kreisen als unsportlich gilt, sollte man sich von dieser Datei ruhig inspirieren lassen. Im Normalfall spart das Zeit.

Inzwischen liegt fast allen Paketen das Tool `configure` bei. Damit wird ausgelesen, was für eine Maschine vorliegt, und die Installation an die lokalen Verhältnisse angepasst. In seltenen Fällen müssen hier auch Parameter angegeben werden. In den meisten Fällen läuft `configure` einfach durch.

Nun werden die Quelltexte übersetzt. Normalerweise startet man dazu den Befehl `make`. Je nach Paket wird vor diesem Aufruf noch ein »`make depend`« verlangt, das die Abhängigkeiten ermittelt. Diese Information befindet sich in der README-Datei. In den allermeisten Fällen wird ein C-Compiler benötigt, am besten der GNU-Compiler. Je nach Software können auch noch `lex`, `yacc`, `Perl` oder andere Entwicklungstools benötigt werden.

Im letzten Schritt wird die Software an die richtigen Stellen gebracht und werden eventuell einige Konfigurationen vorbereitet. Dazu ruft man »`make install`« auf. Eine solche Installation schiebt zwar die Pakete an die richtigen Stellen, hat aber oft keine Deinstallationsmöglichkeit. Einige Pakete bieten allerdings ein `make uninstall` an.

```
configure
make depend
make
make install
```

Solaris Packages

Für Sun Solaris wird Software in Paketen geliefert, die mit den Kommandos `pkgadd` installiert werden können. Alternativ kann man auch das `swmtool` verwenden, das man im `admintool` unter dem Punkt Software ansprechen kann. Da die Installationssoftware protokolliert, welche Dateien installiert worden sind, ist auch eine Deinstallation problemlos möglich.

Mit dem Befehl `pkgadd` wird ein Paket installiert. Mird mit dem Parameter `-d` wird das Verzeichnis angegeben, wo sich die zu installierenden Pakete befinden. Dahinter kann ein spezielles Paket benannt werden.

Der Befehl `pkginfo` liefert eine Liste aller installierten Pakete. Mit der Option `-l` kann ein einzelnes installiertes Paket betrachtet werden. Mit der Option `-ld` erhält man Informationen über ein noch nicht installiertes Paket.

Da Solaris festhält, welche Software wohin installiert wurde, ist es auch in der Lage, installierte Software wieder zu deinstallieren. Dazu dient der Befehl `pkgrm`.

HP-UX: SD-UX

Zentraler Bestandteil einer Software-Distribution für das HP-UX ist das SD-UX, dessen Dämon es ermöglicht, Software netzweit zu installieren und zu verwalten. In einer Standardinstallation ist der `swagentd` nur für eine Maschine zuständig. Dieser Dämon ist der Dreh- und Angelpunkt der Softwareinstallation.

Um Software zu installieren, wird das Kommando `swinstall` verwendet. Der Befehl packt nicht nur das Paket aus, sondern führt auch mitgelieferte Skripten aus, die je nach dem Zeitpunkt ihrer geplanten Ausführung die Namen `preinstall`, `postinstall` und `configure` haben. Für den Fall des Scheiterns, der durch das Skript `checkinstall` festgestellt werden kann, werden auch `unpreinstall` und `unpostinstall` aufgerufen. Als Parameter braucht `swinstall` das Medium oder den Server, auf dem das Softwarepaket liegt und den Namen des Pakets:

```
swinstall -s /dev/rmt/0 pascal
swinstall -s /dev/rmt/0 *
```

Die erste Zeile installiert einen Pascal-Compiler vom Band. Die zweite Zeile installiert alle Pakete, die auf dem Band verfügbar sind. Statt der Kommandozeile kann man auch das Administrationstool `sam` zur Installation von Software verwenden.

Mit dem Befehl `swremove` wird ein Paket wieder entfernt. Die Deinstallation des Pascal-Compilers würde also mit folgendem Befehl erfolgen:

```
swremove -p pascal
```

Schließlich kann man sich mit dem Befehl `swlist` die installierten Pakete anzeigen lassen. An der Ausgabe sieht man, dass das Paket dafür vorbereitet ist, eine netzweite Installation durchzuführen.

```
hpsrv# swlist
# Initializing...
# Contacting target "hpsrv"...
#
# Target:  hpsrv:/
#
```

```
#
# Bundle(s):
#
```

```
B3920BA          B.10.10 HP-UX Media Kit (Reference Only)
  HPUXEngS800    B.10.10 English HP-UX VUE Runtime Environment
hpsrv#
```

Unterabschnitte

- [Informationen holen](#)

Red Hat Package Manager

rpm steht für »Red Hat Package Manager« und hat seinen Ursprung in der Red Hat-Distribution von Linux. rpm verwaltet so genannte Binaries, also nicht nur Quelltextpakete.

Ein rpm-Paket wird zunächst nach dem Namen der Software benannt. Es folgt, durch einen Bindestrich abgetrennt, die Versionsnummer. Ein weiterer Bindestrich trennt die Paketversion ab. Das bedeutet bei höheren Nummern, dass die Pakete neu erstellt worden sind. Es folgen ein Punkt und die Prozessorkennung, für die das Paket generiert wurde, und schließlich die Endung .rpm.

[Wichtige rpm-Kommandos]L|L Kommando & Wirkung

rpm -i Paketname & Installieren eines Pakets

rpm -U Paketname & Paket ersetzen (Update)

rpm -F Paketname & Nur updaten, wenn ein früheres Paket installiert ist.

Informationen holen

Mit der Option -q für Query (engl. Anfrage) kann man einige Informationen über die installierten Pakete bekommen. Sie erscheinen in der Reihenfolge, in der sie installiert wurden:

```
gaston> rpm -q -a
aaa_dir-2001.9.22-0
ash-0.2-395
bash-2.05-82
cpio-2.4.2-413
bdflush-1.5-229
db-3.1.17-109
diffutils-2.7-128
e2fsprogs-1.24a-11
file-3.33-28
gdbm-1.8.0-371
```

[rpm-Anfragen]L|L Kommando & Wirkung

rpm -q -a & Liste aller auf dem Rechner installierten Pakete

rpm -q -l Paketname & Liste aller Dateien des Pakets

rpm -q -i Paketname & Liefert eine Komplettbeschreibung des Pakets

rpm -q -c Paketname & Liste der Konfigurationsdateien

rpm -q -f Dateiname & Ermittelt, zu welchem Paket die Datei gehört

Hier werden als abschließendes Beispiel noch einmal die Informationen gezeigt, die rpm zu einem Programmpaket liefern kann. Man sieht, dass es reicht, den Paketnamen ohne Versionsnummer anzugeben.

```
gaston> rpm -q -i bind
Name       : bind           Relocations: (not relocateable)
Version    : 4.9.7         Vendor: SuSE GmbH, Nuernberg, Germany
Release    : 220           Build Date: Sam 29 Jul 2000 20:49:22
Install date: Fre 30 Nov 2001 12:33:08 CET Build Host: owens.su
Group      : unsorted      Source RPM: bind-4.9.7-220.src.rpm
Size       : 1744974       License: 1989 The Regents of the
University of California.
Packager   : feedback@suse.de
Summary    : Name Server Utilities (old version)
Description:
The named daemon and support utilities including: dig,
dnsquery, host, and nslookup. Documentation on setting up a name
server can be found in /usr/share/doc/packages/bind .
```

Authors:

ISC Software <bind@isc.org>
Paul Vixie <vixie@vix.com>

SuSE series: n
gaston>

« [HP-UX: SD-UX](#) | [Software installieren](#) | [Druckeradministration](#) »

Druckeradministration

Früher wurden Drucker fast ausschließlich über eine serielle Schnittstelle an UNIX-Maschinen angeschlossen. Das hatte mehrere Vorteile. Serielle Leitungen können problemlos 30 m lang sein. Damit brauchte der Drucker nicht direkt im EDV-Raum zu stehen. Man konnte den Drucker mit den gleichen Leitungen versorgen, die man bereits für Terminals verwandte. Die parallelen Anschlüsse waren ursprünglich nur im PC-Umfeld üblich und werden darum in erster Linie von den PC-UNIX-Versionen unterstützt. Allerdings findet man parallele Anschlüsse auch an UNIX-Workstations. Parallele Leitungen sind in ihrer Länge sehr begrenzt, sodass der Drucker dann fast direkt neben dem Computer stehen muss. Inzwischen werden Drucker meistens über eine Netzwerkverbindung angehängt. Netzkabel sind inzwischen in der Regel billiger als serielle Leitungen.

Neben dem reinen Text wünscht man heute auch Formate, Schriften und Grafiken. Dazu braucht man eine Kommandosprache oder Sonderzeichen, die den Drucker anweisen, die Gestaltung vorzunehmen. Unter UNIX ist in diesem Bereich PostScript Standard. PostScript-Drucker sind etwas teurer, dafür aber leicht austauschbar. Dazu kommt, dass man mit der gleichen Datei einen kleinen Laserdrucker wie auch eine Druckmaschine ansteuern kann.

Im PC-Bereich wird scharf kalkuliert, und da PostScript einerseits Lizenzen kostet und andererseits vom Drucker einiges an Speicherausstattung und Rechenleistung fordert, ist PostScript dort selten zu finden. Noch schlimmer ist, dass jeder Druckerhersteller seine eigenen Steuerzeichen verwendet und dass selbst Geräte gleicher Hersteller nicht unbedingt gleich anzusteuern sind. Unter Linux wird darum die freie Software GhostScript verwendet, die PostScript-Dateien für beinahe jeden Drucker übersetzen kann.

Bei den PC-Druckern gibt es seit einigen Jahren Drucker, die auf MS Windows festgelegt sind. Bei diesen so genannten GDI- oder Windowsdruckern wird die Druckaufbereitung im Computer von MS Windows durchgeführt. Der Drucker wird dadurch ein wenig günstiger. Es ist natürlich fast unmöglich, ein solches Gerät unter UNIX zu betreiben. Diese peinliche Kastration tarnen viele Hersteller und deuten sie manchmal nur mit einem W im Namen des Druckers an. Wenn Sie Linux benutzen und auf einen derartigen Drucker bereits hereingefallen sind, lohnt es sich, einmal die Homepage des Herstellers anzusehen. Einige Hersteller bieten Treiber an, mit denen Linux das MS Windows-Umfeld emuliert. Dass das letztlich nicht optimal sein kann, ist offenkundig.

Unterabschnitte

- [Übersicht](#)
- [BSD-Unix: lpd, lpr, lpq und lprm](#)
 - [Die Datei /etc/printcap](#)
 - [Arbeitsablauf des Druckdämons](#)
 - [Drucken über das Netz](#)
 - [Druckjobs administrieren](#)
- [Linux-PC als Druckserver](#)

- [Filter und PostScript](#)
- [Druckerdevices unter Linux](#)
- [Druckerinterrupt auf Linux-Systemen](#)
- [System V: Ipsched, lp, lpstat und cancel](#)
 - [Der Druckdämon Ipsched](#)
 - [Konfiguration per lpadmin](#)
 - [lpstat und cancel](#)
 - [Befehlsübersicht](#)
 - [Installation des BSD-Drucksystems unter SCO](#)
 - [Zugriff auf einen BSD-Netzdrucker unter HP-UX](#)
- [LPRng](#)
- [CUPS - Common UNIX Printing System](#)
 - [Drucker hinzufügen](#)

[« Red Hat Package Manager](#) | [Administration](#) | [Übersicht](#) »

Übersicht

Es gibt zwei Drucksysteme unter UNIX. Dies ist durch die Aufteilung zwischen dem BSD-UNIX aus dem universitären Bereich und dem AT&T-UNIX entstanden. Das AT&T-System gilt als robuster, bietet aber von Haus aus keine Netzwerkunterstützung. Aus diesem Grund hat sich das BSD-System zum Standard entwickelt, wenn es um TCP/IP-Druck geht. Beide Systeme unterscheiden sich auch in den Namen der Befehle. Tabelle zeigt eine kleine Übersicht.

[Kommandounterschiede]

	BSD	AT&T	Funktion
lpd & lpsched	lpd	lpd	Druckdämon
lpr & lp	lpr	lp	Druck absenden
lpq & lpstat	lpq	lpstat	Status der Druckerwarteschlange ermitteln
lprm & cancel	lprm	cancel	Entfernen eines Jobs aus der Druckwarteschlange

Unterabschnitte

- [Die Datei /etc/printcap](#)
- [Arbeitsablauf des Druckdämons](#)
- [Drucken über das Netz](#)
- [Druckjobs administrieren](#)

BSD-Unix: lpd, lpr, lpq und lprm

Die Datei /etc/printcap

Die Datei /etc/printcap ist die Konfigurationsdatei für die Drucker im BSD-System. Hier werden alle von der Maschine erreichbaren Drucker definiert, ihre Warteschlangen und eventuelle Filter. Jeder Drucker wird in einer Zeile beschrieben. Da die Zeilen aber furchtbar lang würden, verwendet man den Backslash am Ende der Zeile. Dieser verlagert das Ende der Zeile auf die nächste Zeile. Die Beschreibung eines Druckers beginnt mit dessen Namen. Ein Drucker kann mehrere Namen haben, die man durch einen senkrechten Strich trennt. Diesen Mechanismus benutzt man auch gern für kurze Druckerbeschreibungen. Üblicherweise nennt man den Standarddrucker des Systems lp.

Durch Doppelpunkte werden Attribute des Druckers abgetrennt. Die wichtigsten Einträge sind das Drucker-Device (lp), das Spoolverzeichnis (sd) und die Protokolldatei (lf). Beipielsweise ist der Standarddrucker auf meinem Linux-System folgendermaßen definiert:

```
lp|lp2|cljet5-a4-auto-color-300|cljet5 a4 auto color 300:
:lp=/dev/lp0:
:sd=/var/spool/lpd/cljet5-a4-auto-color-300:
:lf=/var/spool/lpd/cljet5-a4-auto-color-300/log:
:if=/var/lib/apsfilter/bin/cljet5-a4-auto-color-300:
:mx#0:sh:sf:
```

Der Hauptdruckername ist lp. Er druckt auf die erste parallele Schnittstelle des Computers /dev/lp0. Das Spoolverzeichnis befindet sich unterhalb des Verzeichnisses /var/spool/lpd und wird nach dem Drucker benannt. Darin befindet sich auch die Protokolldatei mit dem Namen log. Der Eintrag hinter if bezeichnet den Druckerfilter, der in diesem Fall durch ein apsfilter-Skript realisiert wird. Der apsfilter sorgt für die Übersetzung der PostScript-Informationen in die druckereigene Steuerungssprache. Hinter dem apsfilter verbirgt sich letztlich GhostScript.

[Parameter der printcap-Datei]L|L Parameter & Bedeutung

mx#0 & maximale Größe des Drucks in Blöcken, 0 heißt unbegrenzt.

sh & Banner abschalten

sf & Form-Feed abschalten

Ein Eintrag für einen Netzwerkdrucker unterscheidet sich von einem lokalen Drucker in erster Linie dadurch, dass das Druckdevice (lp) leer gelassen wird und der Zielhost (rm) und der Druckername auf dem Zielhost (rp) angegeben werden. Ein lokales Spoolverzeichnis ist dennoch notwendig, da der

Druckinhalt zwischengespeichert werden muss, solange der Druckdämon auf der fernen Maschine noch keinen Vollzug meldet. Beispiel:

```
laser|300 dpi Laser|Registratur:  
:lp=:rm=reg_pc01:rp=lp:  
:sd=/var/spool/lpd/laser:
```

Der fremde Rechner heißt reg_pc01 und der Druckername auf jenem Rechner ist lp.

Änderungen in der Datei /etc/printcap liest der Druckdämon lpd beim Start. Der Versuch, den lpd zum Neueinlesen zu bewegen, indem man ihm per kill ein SIGHUP schickt, führt zum spontanen Tod des lpd. Das ist aber normalerweise nicht weiter tragisch, da man ihn einfach neu starten kann. Allerdings ist es durchaus klug, mit dem Abschuss zu warten, bis der lpd gerade keinen Druckauftrag bearbeitet.

Arbeitsablauf des Druckdämons

Der lpd liest beim Start die Datei /etc/printcap. Darin findet er die definierten Drucker und deren Spoolverzeichnisse. Liegen in einem der Spoolverzeichnisse eine Datendatei (beginnt mit d) und eine Kontrolldatei (beginnt mit c) für einen Druckauftrag, geht lpd davon aus, dass dieser Auftrag nicht vollständig ausgedruckt wurde. Ansonsten hätte der vorher laufende lpd ihn ja schließlich gelöscht. Entsprechend stößt er den Druck nun noch einmal an.

Es nutzt also gar nichts, einen nicht gewünschten Ausdruck dadurch zu beenden, dass man den entsprechenden Druckdämon terminiert oder in wilder Panik die Maschine herunterfährt. Im Gegenteil: Der Druck wird wieder ganz von vorn begonnen. Um einen Ausdruck auf brutale Weise abzuwürgen, muss nach dem Terminieren des lpd also auch das Spoolverzeichnis geräumt werden. Mit Hilfe des Befehls fuser (siehe S. fuser) kann man sogar ermitteln, welcher lpd den Druckauftrag bearbeitet, und so einen Ausdruck abbrechen, ohne die anderen Druckaufträge zu gefährden.

Wie man lpr benutzt, um eine Datei zu drucken, wurde im Kapitel Anwendung bereits beschrieben. Nun soll hinter die Kulissen geblickt werden. Das Programm lpr schreibt einen Druckauftrag im Spoolverzeichnis des gewünschten Druckers in eine Datendatei. Danach erzeugt es die Kontrolldatei. Der lpd prüft regelmäßig den Inhalt der Spoolverzeichnisse. Findet er einen Auftrag, druckt er ihn aus, indem er sich an die eingetragene Schnittstelle (Parameter lp) wendet. Ist dieser Eintrag leer und ist stattdessen ein fremder Host (Parameter rm) eingetragen, sendet er beide Dateien an den lpd des entsprechenden Rechners. Er sendet diesem auch die Druckerkennzeichnung (Parameter rp) des Druckers, über den der Zielrechner ausdrucken soll. Schließlich muss der lpd auf dem Zielrechner ja wissen, auf welchem seiner Drucker er ausdrucken soll.

Drucken über das Netz

Empfängt ein lpd einen Druckauftrag über TCP/IP, sieht er in der Datei /etc/hosts.lpd nach, ob der Sender überhaupt berechtigt ist, auf diesem Rechner zu drucken. Die Datei hosts.lpd darf kein öffentliches Schreibrecht haben. Auch alle Rechner, die in der Datei /etc/hosts.equiv stehen, haben Druckrecht. Beim Eintragen des Hostnamens ist darauf zu achten, daß der erste Name des Rechners aus der Datei /etc/hosts verwendet wird, da der Rechner sonst nicht erkannt wird.

Der Druckdämon lpd verwendet zum Senden eines Druckauftrages einen Port (siehe S. port) aus

einem Bereich, den nur ein unter root laufender Prozess erhalten kann. Dies prüft der lpd auf der Gegenseite, um sicher zu sein, dass ihm nicht irgendwer Daten unterschoben will. Passt alles, sendet der Quellrechner die Größe der zu druckenden Datei und die Steuerdaten. Der lpd nimmt sie entgegen. Anschließend werden die Daten übergeben. Der lpd des Zielrechners überprüft noch einmal die Länge und gibt dann sein Ok. Jetzt wird die Spooldatei des Quellrechners gelöscht. Ging irgendetwas schief, wird eine Fehlermeldung gesendet und der Zielrechner löscht alle bisher empfangenen Daten. Die Übertragung beginnt von vorn.

Wenn man den Ablauf dieses Prozesses kennt, wird klar, dass die zigarettenschachtelgroßen Druckserver, die einen gewöhnlichen Drucker zum TCP/IP-Drucker aufwerten sollen, schwerlich das Protokoll einhalten können. An sich müssten sie die gesamten Daten zwischenspeichern, bis sie sicher sind, dass die Übertragung erfolgreich war. Da sie keine Festplatte besitzen, müssen sie den Kompromiss eingehen, dass sie vorab von einer korrekten Übertragung ausgehen und bereits ausdrucken, bevor die Daten komplett und bestätigt übertragen wurden.

Druckjobs administrieren

Die Druckjobverwaltung erfolgt über die Befehle lpq und lprm. lpq liefert für den angegebenen Drucker Informationen über alle anstehenden Druckjobs.

```
gaston# lpq
lp is ready and printing
Rank  Owner      Job  Files                Total Size
active arnold    97   unix.ps            4591488 bytes
```

Diese Meldung besagt, dass auf dem Drucker lp ein Job mit der Nummer 97 aktiv ist, den der Benutzer arnold gestartet hat. Die in Auftrag gegebene Datei heißt unix.ps und ist etwa 4,5 MByte groß.

Mit dem Befehl lprm 97 kann der oben stehende Druckjob wieder entfernt werden. Die Ausgabe des Befehls sieht etwa so aus:

```
gaston# lprm 97
dfA097gaston dequeued
cfA097gaston dequeued
```

Hier wird deutlich, dass ein Druck aus zwei Dateien besteht. Dies ist einmal die Steuerdatei, deren Name mit cf (control file) beginnt und zum anderen die Datendatei, deren Name mit df (data file) beginnt. Beide Dateien befinden sich im Spoolverzeichnis.

Um einen Drucker oder seine Warteschlange aus dem System zu entfernen, verwendet man den Befehl lpc. Dieser Befehl hat mehrere Kommandos als Parameter. Der wichtigste Parameter des Befehls ist der Name des Druckers.

[lpc-Kommandos]L|L Befehl & Wirkung

lpc disable lp & Sperre die Warteschlange für lp

lpc enable lp & Gib die Warteschlange für lp frei

lpc stop lp & Stoppe das Drucken aus der Warteschlange für lp

lpc start lp & Lasse das Drucken aus der Warteschlange für lp wieder zu

lpc down lp & Stoppe das Drucken und die Warteschlange für lp

lpc up lp & Starte das Drucken und die Warteschlange für lp wieder

lpc status lp & Informationen über den Zustand der Sperrungen

Mit dem Befehl `lpc` können auch anstehende Druckaufträge in ihrer Reihenfolge verändert werden. Dazu werden das Kommando `topq`, der Drucker und die Jobnummer angegeben, die an die erste Stelle gesetzt werden soll.

```
gaston# lpc topq lp 97
```

« [Übersicht](#) | [Druckeradministration](#) | [Linux-PC als Druckserver](#) »

Unterabschnitte

- [Filter und PostScript](#)
 - [Druckerdevices unter Linux](#)
 - [Druckerinterrupt auf Linux-Systemen](#)
-

Linux-PC als Druckserver

Ein PostScript-Drucker mit Netzchnittstelle ist nicht billig. Da ist es ein nahe liegender Gedanke, einen ausgedienten PC mit den preiswerten PC-Druckern auszustatten und an das Netz anzuschließen. Gegenüber einem netzwerkfähigen TCP/IP-Drucker hat diese Lösung den Vorteil, dass der Linux-PC das lpd-Protokoll korrekt einhält. Die PostScript-Fähigkeit wird durch GhostScript realisiert, und als Zugabe kann man die Drucker sogar in Samba (siehe S. samba) einbinden und dadurch von den Arbeitsplätzen unter MS Windows aus mitverwenden.

Filter und PostScript

Unter UNIX ist PostScript der Standard für Drucker. Das ist sehr praktisch, da die meisten Anwendungsprogramme kein Problem damit haben, eine PostScriptausgabe zu erzeugen. Das Programm GhostScript ist ein Übersetzer von PostScript auf die verschiedensten Ausgabemedien. Dazu zählen neben dem Bildschirm auch alle gängigen Drucker.

Die Umwandlung von PostScript in die druckereigene Sprache erfolgt durch einen Filter, der mit dem Eintrag `if` in die Datei `/etc/printcap` eingetragen wird. Bei der Linux-Distribution von S.u.S.E. erscheint dort nach einer Druckerinstallation ein `apsfilter`. Dahinter verbirgt sich ein Programm, das den Programmnamen durch einen Aufruf von `gs` ersetzt. GhostScript setzt nun den Input in PostScript auf diverse proprietäre Formate um.

Ein Interface-Skript wie `apsfilter` kann man unter anderen Linuxsystemen leicht selbst schreiben, indem man eine Datei mit folgender Zeile erzeugt: vgl. Barakati, Naba: Linux Red Hat 6.0, 2000, Franzis', S. 466

```
/usr/bin/gs -q -dNOPAUSE -dSAFER -sDEVICE=clj5 -soutputFile=-
```

Der Name des Skripts (beispielsweise `/usr/local/cljf`) wird dann unter `:if=/usr/local/cljf:` in die `/etc/printcap` eingetragen. Der unter `if` eingetragene Filter erhält die Eingabedaten über `stdin` und muss die Ausgabedaten nach `stdout` wieder abliefern. Das ist der Grund für den Bindestrich hinter `-soutputFile=`.

Druckerdevices unter Linux

Der Zugriff auf die Schnittstellen erfolgt über die Gerätedateien im Verzeichnis `/dev`, wie dies unter UNIX üblich ist. Bei PCs sind die Druckeranschlüsse parallele Schnittstellen. Davon kann jeder PC drei Stück haben, wobei der dritte keinen Interrupt besitzt und per Polling abgefragt werden muss. Dazu

kommen die USB-Anschlüsse und die maximal vier Mit einer speziellen Karte sind auch mehr als vier serielle Schnittstellen möglich. Diese teilen sich dann den Interrupt. Man sollte aber vor der Verwendung prüfen, ob sie auch von Linux unterstützt werden. seriellen Schnittstellen, von denen wiederum zwei mit eigenen Interrupts belegbar sind. Insgesamt kann man einen solchen Rechner also recht ordentlich bestücken.

[Druckeranschlüsse am PC]L|C Device & Anschluss & Interrupt
/dev/lp0 & parallele Schnittstelle (LPT1) & 7
/dev/lp1 & parallele Schnittstelle (LPT2) & 5
/dev/ttyS0 & serielle Schnittstelle (COM1) & 3
/dev/ttyS1 & serielle Schnittstelle (COM2) & 4
/dev/usb/lp0 & per USB angeschlossener Drucker &

Druckerinterrupt auf Linux-Systemen

Auf einem Einzelplatzsystem wie MS-DOS wartet das Programm normalerweise in einer Endlosschleife auf den Druckauftrag. Dies nennt man Polling. Die Alternative dazu ist die Interruptsteuerung. Dazu wird der Prozess, der den Druck steuert, direkt nach Ablieferung eines Druckzeichens in eine Warteschlange gesetzt. Die CPU wendet sich neuen Aufgaben zu. Sobald der Drucker fertig ist, unterbricht er (Interrupt) die derzeitige Beschäftigung der CPU. Diese holt den wartenden Prozess wieder aus der Warteschlange und setzt ihn fort.

Während das Polling zu einer Beobachtungsschleife führt, die die CPU stark belastet, findet bei der Interruptsteuerung nur dann eine Aktion statt, wenn der Ablauf das erfordert. Bei Einzelplatzsystemen ist der Unterschied nicht signifikant. Unter Multitasking-Systemen, insbesondere bei mehreren Druckern, sollte eine interruptgetriebene Ansteuerung verwendet werden.

Beim Design des PCs waren von vornherein Interrupts für den Drucker vorgesehen. Für die erste parallele Schnittstelle ist der Interrupt 7 reserviert. Ein zweiter Parallelanschluss (LPT2) ist mit dem Interrupt 5 vorgesehen. Aufgrund der Interrupt-Knappheit in einem PC wird dieser Interrupt allerdings häufig für andere Peripherie verwendet, da nur wenige PCs wirklich zwei Druckerschnittstellen besitzen. Ein dritter Anschluss (LPT3) ist zwar ansteuerbar, hat aber keinen Interrupt zugeordnet.

Unter Linux kann durch den Befehl `tunelp` der Druckerinterrupt eingeschaltet und zugeordnet werden. Standardmäßig arbeitet LINUX mit Polling. Interessanterweise erreicht Linux auch so eine ganz anständige Leistung aus dem Drucker. Wenn aber Leistungseinbrüche beim Druck oder hohe Belastungen des Systems durch den Druck feststellbar sind, sollte man die Interrupts aktivieren.

```
tunelp /dev/lp1 -i7  
tunelp /dev/lp2 -i5
```

Hier wird die Schnittstelle LPT1 auf den Interrupt 7 und die Schnittstelle LPT2 auf den Interrupt 5 gesetzt. Seit der Kernelversion 2.1.131Die Kernelversion meldet der Befehl `uname -r`. wird die Steuerung der parallelen Schnittstellen durch das `parport`-System übernommen. In der Konfigurationsdatei `/etc/modules.conf` erreicht der folgende Eintrag, dass die Druckerschnittstellen 1 und 2 per Interrupt betrieben werden.

```
alias parport_lowlevel parport_pc  
options parport_pc io=0x378,0x278 irq=7,5
```

Unterabschnitte

- [Der Druckdämon Ipsched](#)
 - [Konfiguration per lpadmin](#)
 - [lpstat und cancel](#)
 - [Befehlsübersicht](#)
 - [Installation des BSD-Drucksystems unter SCO](#)
 - [Zugriff auf einen BSD-Netzdrucker unter HP-UX](#)
-

System V: Ipsched, lp, lpstat und cancel

Das Drucksystem von System V ist bei den neueren Systemen von Solaris, bei HP-UX, AIX, SCO und anderen Systemen das Standardverfahren zum Drucken. Es bietet einige sehr interessante Eigenschaften. Allerdings hat man leider kein netzweites Drucken vorgesehen. Das führt zu Erweiterungen wie bei HP. Dort ist es möglich, einen Netzdrucker nach BSD-Standard anzusteuern. Andere Systeme, wie SCO, erlauben es, stattdessen das BSD-System zu installieren.

Der Druckdämon Ipsched

Der Druckdämon des System V-Drucksystems heißt `lpsched` und wird durch die rc-Dateien beim Systemstart hochgefahren. Der Dämon verhindert einen versehentlichen Doppelstart, indem er eine Sperrdatei namens `SCHEDLOCK` anlegt, die im Verzeichnis `/var/spool/lp` liegt. Normalerweise wird der Dämon durch den Aufruf von `lpshut` wieder gestoppt. Sollten Sie in die Verlegenheit kommen, den Server auf andere Weise zu beenden, müssen Sie vor einem Neustart die Sperrdatei von Hand löschen.

Konfiguration per lpadmin

Das Administrationstool von System V-Drucksystemen heißt `lpadmin`. Auf HP-UX-Systemen wird vor der Verwendung von `lpadmin` der Dämon mit `lpshut` heruntergefahren. Bei Solaris dagegen muss der Dämon `lpsched` laufen, damit `lpadmin` funktioniert.

Mit dem Tool `lpadmin` werden in erster Linie Drucker eingerichtet. Das folgende Beispiel richtet einen Drucker namens `laser` ein, der auf der Gerätedatei `/dev/lp` erreichbar ist. Es handelt sich um einen Farblaserdrucker vom Modell `colorlaserjet`.

```
lpadmin -plaser -v/dev/lp -mcolorlaserjet
```

Bei Einrichten eines neuen Druckers muss mindestens der Druckername, also die Option `-p`, die Gerätedatei (`-v`) und der Druckertyp angegeben werden. Der Druckertyp wird im oberen Beispiel mit dem Modell (`-m`) bestimmt. Welche Modelle ein System kennt, ist an den Dateien im Verzeichnis `/var/spool/lp/model` abzulesen. Am portabelsten ist natürlich das Modell `postscript`. Alternativ kann mit `-i` ein Skript angegeben werden, das die Umformatierung vornimmt. Dabei ist es im Gegensatz zum `i f`-Parameter in der `printcap` des BSD-Systems kein Filterprogramm. Es wird vom `lpsched` aufgerufen

und erhält die auszugebenden Dateinamen als Parameter. Die Argumente sind im Einzelnen: vgl. Nemeth, Evi / Snyder, Garth / Seebass, Scott / Hein, Trent R.: UNIX Systemverwaltung. Markt+Technik - Prentice Hall, München, 2001. S. 857.

AuftragsID Benutzer Titel Kopien Optionen Datei(en)

Die formatierten Inhalte gibt das aufgerufene Skript über die Standardausgabe aus.

Mit der Option `-e` kann der Druckertyp auf einen bereits existierenden Drucker bezogen werden.

Eine Besonderheit des System V-Druckdienstes sind die Druckerklassen. Damit können mehrere Drucker gleichzeitig als Ziel eines Ausdrucks benannt werden. `lpsched` wird denjenigen Drucker aus der Klasse wählen, der derzeit frei ist. Durch die Option `-c` wird beim `lpadmin` angegeben, dass der bezeichnete Drucker einer bestimmten Klasse zugeordnet wird. Um beim Drucken eine Klasse statt einem Druckziel anzugeben, wird der Befehl `lp` mit dem Parameter `-c`, gefolgt vom Klassennamen aufgerufen.

Mit dem Befehl `lpadmin` kann auch festgelegt werden, welcher der definierten Drucker der Standarddrucker des Systems sein soll. Dazu wird `lpadmin` mit der Option `-d` und dem Namen des Druckers aufgerufen.

Das Löschen eines Druckers erfolgt mit der Option `-x` des `lpadmin`. Ist der Drucker der letzte seiner Klasse, wird auch die Klasse gelöscht. Soll ein Drucker nur geändert werden, wird der Drucker aufgerufen, als würde er neu angelegt. Es müssen aber nur die Optionen angegeben werden, die sich ändern sollen.

Die Optionen von `lpadmin` im Überblick:

- `[-pDruckername]`
Dies ist der Name, unter dem der Drucker den Druckbefehlen bekannt sein soll.
- `[-vGerätefile]`
Das ist das Device, an dem der Drucker angeschlossen ist. Es ist aber nicht zwingend, dass es sich um ein Gerät handelt. Es ist durchaus möglich, mit Hilfe des Drucksystems Ausgaben in eine Datei zu leiten, bei denen das System dafür sorgt, dass sie nacheinander geschrieben werden.
- `[-eDruckername]`
Das Druckermodell des unter `-p` angegebenen Druckers ist das gleiche wie das des bereits existierenden Druckers, der hier mit `-e` benannt wird.
- `[-mModell]`
Der Druckertyp entspricht dem eines dem System bereits bekannten Modells. Die jeweils bekannten Modelle finden sich im Verzeichnis `/var/spool/lp/model`.
- `[-iSkript]`
Die Anpassung an den Druckertyp erfolgt durch das angegebene Skript.
- `[-xDruckername]`
Der Drucker wird aus der Liste der verfügbaren Drucker gelöscht.
- `[-dDruckername]`
Richtet den Drucker als Standarddrucker ein.
- `[-cKlasse]`
Ordnet den Drucker der genannten Klasse zu.
- `[-rKlasse]`
Nimmt den Drucker aus der genannten Klasse. Ist damit die Klasse leer, wird die Klasse entfernt.

`lpadmin` erzeugt und verändert Textdateien, die unter dem Verzeichnis `/var/spool/lp` stehen. Es empfiehlt sich nicht, diese Dateien mit einem Editor zu bearbeiten.

lpstat und cancel

Der Befehl `lpstat` gibt eine Liste über den Status der verschiedenen Drucker aus.

Queue	Dev	Status	Job	Files	User	PP %	Btks	Cp	Rnk
lp0	lp0	RUNNING	918	STDIN.14846	willemer	5 43	10	1	1
		QUEUED	919	STDIN.14593	willemer		3	1	2
		QUEUED	920	maskexpj	wagener		2	1	3
		QUEUED	921	STDIN.14596	willemer		2	1	4
		QUEUED	922	STDIN.21790	willemer		1	1	5
		QUEUED	923	lunget	wagener		3	1	6
		READY							
bsh	bshde								

Um einen Druckauftrag aus der Warteschlange zu entfernen, wird der Befehl `cancel` verwendet. Als Argument wird die Jobnummer verwendet. Jeder Anwender darf nur eigene Druckaufträge entfernen, `root` darf jeden beliebigen Druckjob entfernen. Ist ein Drucker durcheinander geraten, erkennt man dies bei der Ausgabe von `lpstat` am Status `DOWN`. Um diesen Drucker nach der Beseitigung der Störung wieder in Betrieb zu nehmen, wird das Kommando `enable` mit der Druckerbezeichnung als Argument verwendet. Soll ein Drucker dagegen gesperrt werden, wird der Befehl `disable` mit dem Druckernamen als Parameter verwendet.

Befehlsübersicht

Tabelle zeigt die Druckbefehle des Systems V noch einmal in der Übersicht.

- [Übersicht über die Druckerbefehle von System V] | L Befehl & Wirkung
- lp sched & Druckdämon
- lpshut & Herunterfahren des Dämons
- lpstat & Zeigt den Zustand der Warteschlangen der Drucker an
- cancel [Jobnr] [Drucker] & Druckjob aus der Warteschlange entfernen
- disable Drucker & Drucker aus dem Drucksystem auskoppeln
- enable Drucker & Drucker in das Drucksystem einkoppeln

Installation des BSD-Drucksystems unter SCO

SCO hat standardmäßig das System V-Druckprotokoll installiert. Um mit einem System auf `lpd`-Basis arbeiten zu können, muss das BSD-System über das bisherige installiert werden. SCO bezeichnet das BSD-`lpd`-System als Remote-Line-Printing (`rlp`). Die Installation geschieht über den Befehl:

```
mkdev rlp
```

Zugriff auf einen BSD-Netzdrucker unter HP-UX

HP-UX hat den Druckserver dahingehend erweitert, dass man mit dem Befehl `lpadmin` auch Netzdrucker erreichen kann. Als Netzprotokoll wird das Protokoll des BSD `lpd` verwendet. HP-UX

sendet den Druckauftrag an einen lpd-Server, wenn beim Befehl `lpadmin` hinter der Option `-m` das Wort »rmodel« steht. Dann kann mit Hilfe weiterer Optionen bestimmt werden, an welchen Drucker der Auftrag übermittelt wird. Die benötigten Parameter lauten:

- `[-m]` muss auf `rmodel` stehen.
- `[-ormHostname]` steht für den Zielrechner.
- `[-orpPrinter]` ist der Druckername auf dem Zielsystem.
- `[-ob3]` verwendet dreistellige Anfragenummern, die für BSD-Zugriffe gebraucht werden.

Um den Drucker `lp` vom Rechner `gaston` als Drucker `linux` auf HP-UX einzurichten, wird folgende Befehlsfolge eingegeben.

```
hp# lpshut
scheduler stopped
hp# lpadmin -plinux -v/dev/null -mrmodel -ormgaston -orp1p -ob3
hp# accept linux
destination "linux" now accepting requests
hp# enable linux
printer "linux" now enabled
hp# lpsched
scheduler is running
hp# lp -d linux /etc/hosts
request id is linux-0 (1 file)
hp#
```

Der letzte Befehl ist lediglich der Test, ob der Druck funktioniert.

« [Linux-PC als Druckserver](#) | [Druckeradministration](#) | [LPRng](#) »

LPRng

Die Existenz zweier Drucksysteme ist natürlich wenig befriedigend, und es ist eigentlich ein nahe liegender Gedanke, beide zusammenzuführen und sowohl die Kommandos des einen als auch des anderen Systems zuzulassen. LPRng ist eine solche Neuentwicklung, die versucht, das Beste des BSD- und des System V-Drucksystems zu einem System zusammenzufassen und an einigen Stellen zu modernisieren.

LPRng ist eine Neuimplementation des BSD-Drucksystem und hält sich vollständig an die Schnittstellen des RFC 1179 und erbt damit die Netzwerkfähigkeit des lpd. LPRng verwendet wie das BSD-System die Aufruf `lpr`, `lprm` und auch die anderen Kommandos des BSD-Systems. Auch die `/etc/printcap` kann direkt weiterverwendet werden. Hinzu kommen die Befehle des Systems V, die als Links ausgeprägt sind, sodass beide Arten, den Druckprozess anzusprechen, funktionieren. Darüber hinaus beherrscht LPRng auch die dynamische Druckzuteilung, also das, was unter System V Druckerklassen sind. Über beides hinaus geht die wesentlich bessere Fehlerinformation des Aufrufers.

Neben der Datei `/etc/printcap` gibt es die Datei `lpd.conf` und die Datei für die Festlegung der Berechtigungen namens `lpd.perms`. Jeder Benutzer kann sich eine eigene Datei `.printcap` definieren, die er in seinem Heimatverzeichnis abstellt. Diese Datei wird vom System bei der Interpretation der systemweiten `/etc/printcap` quasi vorangestellt. Dadurch kann jeder Benutzer festlegen, welcher Drucker sein Standarddrucker ist und natürlich eigene Drucker hinzufügen. Die Datei `lpd.conf` kann normalerweise so bleiben, wie sie als Standard ausgeliefert wird. Auch die `lpd.perms` kann so bleiben wie sie ist, wenn man nicht besondere Benutzer zulassen oder aussperren will. Zu beiden gibt es eine eigene Manpage.

Zu LPRng gehört das Programm `checkpc`, das die Installationsdateien prüft. Sie sollten das Programm nach Änderungen in den Konfigurationsdateien aufrufen, um sicherzugehen, dass die Konfiguration stimmt. Das Programm wird bei einigen Linux-Distributionen in die `rc`-Datei gestellt. Leider blockiert dieses Programm unter ungünstigen Umständen und verhindert in diesem Fall den Systemstart. Aus diesem Grund sollten Sie den Aufruf auskommentieren. Man findet ihn, wenn man `grep checkpc` in dem Verzeichnis aufruft, in dem die `rc`-Skripten stehen.

Unterabschnitte

- [Drucker hinzufügen](#)

CUPS - Common UNIX Printing System

Das Common UNIX Printing System (CUPS) ist ein Paket der Firma Easy Software Products, das verschiedene Designprobleme der alten Drucksysteme beseitigen will. Es ist auf den wichtigsten UNIX-Systemen bereits verfügbar und wurde unter die GNU-Lizenz gestellt, ist also freie Software.

Nach dem Installieren des Paketes kann man normalerweise bereits den Server von CUPS starten. Er heißt `cupsd` und befindet sich im Verzeichnis `/usr/sbin`.

CUPS unterstützt sowohl die BSD-Kommandos als auch die von System V. Die Druckerklassen von System V kennt CUPS genauso wie die Fähigkeit von BSD zum Netzbetrieb.

Drucker hinzufügen

Die Administration des CUPS kann weitgehend über einen Browser erfolgen. Dabei wird allerdings kein Webserver mit dem Standardport 80 aufgerufen, sondern der CUPS-Server, der über den Port 631 erreichbar ist.

```
http://localhost:631/admin
```

Alternativ kann mit dem Kommandozeilenwerkzeug `lpadmin` gearbeitet werden. Die Option `-p` fügt einen Drucker hinzu. Die allgemeine Form sieht so aus:

```
/usr/sbin/lpadmin -p printer -E -v device -m ppd
```

Ein paar Beispiele beleben die Vorstellungskraft. Soll ein HP Deskjet an den parallelen Port des Computers angeschlossen werden, lautet der Aufruf:

```
lpadmin -p DeskJet -E -v parallel:/dev/lp1 -m deskjet.ppd
```

Ein HP Laserjet mit einer JetDirect-Netzwerkkarte mit der IP-Adresse 192.168.109.192 würde mit dem folgenden Kommando eingerichtet:

```
lpadmin -p HPLJ -E -v socket://192.168.109.192 -m laserjet.ppd
```

Im Verzeichnis `/etc/cups` finden sich die Konfigurationsdateien von CUPS, insbesondere die Datei `cupsd.conf`. Ihr Aufbau erinnert fast bis ins Detail an die Konfigurationsdatei `httpd.conf` des Webservers `apache` (siehe S. `apache`).

Beispielsweise kann der Zugriff auf die Drucker mit der `Allow`-Anweisung wie bei der Konfiguration des

apache auf einzelne Rechner oder bestimmte Netze beschränkt werden.

Das System zeichnet sich besonders dadurch aus, dass es per Browser administriert wird. Das hat natürlich den Vorteil, dass kein Portierungsaufwand bei der grafischen Oberfläche erforderlich ist und dass die Administration gleichzeitig per Netz erfolgen kann.

« [LPRng](#) | [Druckeradministration](#) | [Terminals](#) »

Terminals

Terminals bestehen aus einer Einheit aus Tastatur und einem Monitor, die unter UNIX normalerweise über eine serielle Leitung mit dem Rechner verbunden sind. Dabei besitzen die einfachen Terminals nur die Fähigkeit, mit festen Zeichen umzugehen. Da es Terminals verschiedener Hersteller gibt, liefern Terminals je nach Typ unterschiedliche Codierungen für die Sondertasten und benötigen unterschiedliche Sequenzen, um den Bildschirm anzusteuern.

Echte Terminals gibt es kaum noch. Da Netzverkabelungen heute billiger sind als serielle Kabel und die PCs sowieso auf jedem Arbeitsplatz stehen, gehen immer mehr Terminals in Rente. An einer Stelle allerdings sind sie in manchen Fällen von unschätzbarem Wert: im Serverraum als Konsole. Denn im Falle eines Zusammenbruchs der Netzwerkkomponente ist es gut, einen Zugang zur Maschine zu haben, der nicht vom Netzwerk abhängig ist.

Unterabschnitte

- [Konfiguration der Terminals](#)
 - [Die Terminalvariable TERM](#)
 - [termcap](#)
 - [terminfo](#)
 - [Wenn das Terminal durcheinander ist](#)
-

Konfiguration der Terminals

Bei Hochfahren des Systems liest der Prozess `init` die Datei `/etc/ttys` (bei BSD-Systemen) oder `/etc/inittab` (bei System V), um festzustellen, welche Terminals verfügbar sein sollen. In beiden Systemen existiert für jedes Terminal eine Zeile.

Die Datei `/etc/ttys` unter BSD ist recht einfach aufgebaut. Zunächst kommt der Gerätenamen des Terminals, es folgt der Programmaufruf des `getty`, der für dieses Terminal zuständig ist. Anschließend kommt der Terminaltyp der `termcap`, der für dieses Terminal standardmäßig verwendet werden soll. Die beiden folgenden Spalten geben an, ob man sich als `root` anmelden kann oder nicht.

<code>console</code>	<code>none</code>	<code>unknown</code>	<code>off</code>	<code>secure</code>
<code>ttyv0</code>	<code>"/usr/libexec/getty Pc"</code>	<code>cons25</code>	<code>on</code>	<code>secure</code>
<code>ttyv1</code>	<code>"/usr/libexec/getty Pc"</code>	<code>cons25</code>	<code>on</code>	<code>secure</code>
<code>ttyv2</code>	<code>"/usr/libexec/getty Pc"</code>	<code>cons25</code>	<code>on</code>	<code>secure</code>
<code>ttyv3</code>	<code>"/usr/libexec/getty Pc"</code>	<code>cons25</code>	<code>on</code>	<code>secure</code>
<code>ttyv4</code>	<code>"/usr/libexec/getty Pc"</code>	<code>cons25</code>	<code>on</code>	<code>secure</code>

Der Eintrag in der `inittab` sieht geringfügig anders aus, erfüllt aber den gleichen Zweck:

<code>5:123:respawn:/sbin/mingetty tty5</code>
<code>6:123:respawn:/sbin/mingetty tty6</code>
<code>S0:123:respawn:/sbin/agetty -L 9600 ttyS0</code>

Ein Eintrag in der `inittab` hat folgende Struktur:

ID:Runlevel:Aktion:Prozess

- [ID:] Dies ist die Kennung der Zeile, sie besteht aus maximal zwei oder vier Zeichen.
- [Runlevel:] Legt fest, in welchem Runlevel die Aktion auszuführen ist. Im Beispiel sind die Aktionen in Runlevel 1, 2 und 3 aktiv.
- [Aktion:] Hier können verschiedene Schlüsselwörter stehen. Im Falle eines Terminaleintrags steht hier immer `respawn`. Das bedeutet, dass sofort ein neuer Prozess gestartet wird, wenn der Prozess stirbt.
- [Prozess:] Der Prozess, der zu diesem Eintrag gehört und gestartet wird.

Um ein weiteres Terminal anzuschließen, ist ein neuer Eintrag notwendig. Anschließend muss der Prozess `init` dazu gebracht werden, die Datei erneut zu lesen. Man erreicht dies, indem man `init` das `SIGHUP`-Signal zusendet. Der Befehl lautet:

<code>kill -1 1</code>

Der `init`-Prozess startet beim Hochfahren für jede Terminalleitung den entsprechenden Befehl, so wie er in `/etc/inittab` bzw. `/etc/ttys` aufgeführt ist. Normalerweise ist dies `getty` oder ein verwandtes Programm. `getty` überwacht die Leitung. Tut sich etwas, startet er den Befehl `login`, der wiederum bei erfolgreicher Anmeldung die Shell startet. Beim Abmelden wird die Shell einfach beendet.

Ebenfalls in der `inittab` steht, was nach dem Tod des letzten Programms passieren soll. Für Terminals steht hier `respawn`, da nach der Abmeldung bzw. der erfolglosen Anmeldung `getty` wieder in Aktion treten soll.

Es ist bei vielen Systemen möglich, mehrere Baudraten für ein Terminal anzugeben. Eine falsche Baudrate zeigt sich am Terminal durch wilde Zeichen. Durch Drücken der Taste `Break` in den meisten Terminalemulationen findet sich in den Menüs ein Kommando, um einen Break zu senden. am Terminal bringt man `gpverb+getty+` dazu, zyklisch die nächste Baudrate zu verwenden. Man drückt also `Break`, bis man das Login lesen kann, und meldet sich dann an.

Eine Anmeldung nur in Großbuchstaben interpretiert `getty` so, dass das Terminal nur Großbuchstaben beherrscht. Darum werden alle Großbuchstaben als Kleinbuchstaben interpretiert. Echte Großbuchstaben werden mit einem voranstehenden Backslash erzeugt. Da es heute keine Terminals mehr gibt, die keine Kleinschreibung beherrschen, hat man wahrscheinlich versehentlich die Caps-Lock-Taste bei der Anmeldung gedrückt. Meldet man sich einfach erneut an, ist alles wieder im Normalzustand.

« [Terminals](#) | [Terminals](#) | [Die Terminalvariable TERM](#) »

Die Terminalvariable TERM

Das eingestellte Terminal ist in der Umgebungsvariablen TERM hinterlegt. Für kurzfristige Änderungen des Terminals kann man den Inhalt dieser Variablen verändern. Da Umgebungsvariablen nicht automatisch an die Kindprozesse weitervererbt werden, sollte TERM exportiert werden. Greift dies nicht, kann man das Terminal mit dem Aufruf von `tset` initialisieren.

```
TERM=vt100
export TERM
tset
```

Das aktuelle Terminal wird auf vt100 eingestellt.

termcap

In den BSD-Varianten wie beispielsweise Solaris oder FreeBSD werden in der Datei /etc/termcap (Terminal Capabilities) die Charakteristika der verschiedenen Terminals mit den zugehörigen Steuersequenzen definiert. Diese Datei hat gewisse Analogien zur Datei /etc/printcap (siehe S. printcap).

Als Beispiel ist hier der Eintrag für ein VT52-Terminal herausgegriffen, da dieses nicht so kompliziert ist. Es kennt keine Funktionstasten oder Farbe. An diesen Einträgen kann man bereits ermessen, dass das Einrichten eines Terminals schnell zur Geduldsprobe werden kann. Wenn Sie also eine eigene Terminalanpassung brauchen, kopieren Sie sich ein möglichst ähnliches Terminal, und führen Sie die notwendigen Anpassungen durch.

```
vt52|dec vt52:
:bs:
:co#80:it#8:li#24:
:ac=``aaffggjjkkllmmnnnooppqqrrssttuuvvwxxyyzz{{||}} :
:ae=EG:as=EF:bl=^G:cd=EJ:ce=EK:cl=EHEJ:cm=EY%+ %+ :
:cr=^M:do=EB:ho=EH:kb=^H:kd=EB:kl=ED:kr=EC:ku=EA:
:le=ED:nd=EC:nw=^M^J:sf=^J:sr=EI:ta=^I:up=EA:
```

Es gibt drei Arten von Variablen. Boolesche Variablen werden wahr, wenn sie aufgeführt werden. Nicht genannte Variablen sind immer falsch. Numerische Variablen verwenden ein Hashzeichen (#), und Zeichenfolgen werden mit einem Gleichheitszeichen gekennzeichnet. Bei den Zeichenketten wird das ESC-Zeichen als E notiert. Kontrollzeichen wird ein ^ vorangestellt. Die Bedeutung der wichtigsten Variablen ist in den Tabellen und aufgeführt.

[Bildschirmeigenschaften]C|L Kürzel & Bedeutung

bs & Backspace führt Rückschritt aus
co# & Anzahl der Spalten
li# & Anzahl der Zeilen
it# & Tabulatorenweite
as= & Startkennung des alternativen Zeichensatzes
ac= & Zeichenpaare für Blockgrafik
ae= & Endekennung des alternativen Zeichensatzes
bl= & Zeichen, das einen Piep auslöst (bell)
cd= & Löschen bis Bildschirmende
ce= & Löschen bis Zeilenende
cl= & Bildschirm komplett löschen, Cursor links oben
ho= & Cursor eine Zeile nach links oben positionieren
cm= & Cursor positionieren
do= & Cursor eine Zeile nach unten bewegen
le= & Cursor ein Zeichen nach links bewegen
nd= & Cursor ein Zeichen nach rechts bewegen
up= & Cursor eine Zeile nach oben bewegen

[Tastatureigenschaften]C|L Kürzel & Bedeutung

cr= & Eingabezeichen für Wagenrücklauf

kb= & Taste für Backspace
kd= & Taste für Cursor unten
kl= & Taste für Cursor links
kr= & Taste für Cursor rechts
ku= & Taste für Cursor oben
nw= & Kommando Carriage Return
sf= & Eine Zeile scrollen
sr= & Rückwärts scrollen
ta= & Nächsten Tabulator anspringen

Weitere Variablenkürzel finden Sie in der Manpage von termcap.

« [Die Terminalvariable TERM](#) | [Terminals](#) | [terminfo](#) »

terminfo

System V und Linux verwenden die Datei `/etc/terminfo` zur Ansteuerung von Terminals. Diese Datei liegt nicht mehr als eine große Textdatei wie die `termcap` vor. Durch die vielen unterschiedlichen Terminals ist eine `termcap` derart groß, dass die Verwaltung schwierig wird.

Der Eintrag eines Terminals wird in einer eigenen Textdatei erstellt, dann mit dem Terminfo-Compiler `tīc` kompiliert und anschließend unter dem Verzeichnis `/usr/lib/terminfo` abgestellt.

Um eine Terminalbeschreibung zu finden, beginnt man mit der Terminalbezeichnung. Der Name (`vt100`, `wyse370` etc.) führt zu einer Datei unterhalb des Verzeichnisses `/usr/lib/terminfo`. Dort finden sich Verzeichnisse mit einem Buchstaben. Diese Buchstaben sind der jeweils erste Buchstabe der Terminalbezeichnung. Auf diese Weise wird das Verzeichnis `terminfo` nicht überfüllt, was ansonsten zu einer langen Antwortzeit führen würde. Um auf die Beschreibung eines Terminals mit der Kennung `vt100` zuzugreifen, sucht UNIX die Datei `vt100` in dem Verzeichnis `/usr/lib/terminfo/v`.

Diese Datei liegt wie gesagt im Binärformat vor, ist also nicht direkt lesbar. Um eine textuelle Darstellung der Terminfodatei zu erhalten, verwendet man den Befehl `infocmp`. Durch Umleiten der Ausgabe kann man eine Sourcedatei erstellen. In dieser Datei werden die Eigenschaften des Terminals beschrieben. Sie können geändert werden und mit `tīc` wieder zu Terminfodateien kompiliert werden.

Die exakte Beschreibung der Einträge in `terminfo` sowie die Funktionsweise von `tīc` und `infocmp` können den jeweiligen Manpages entnommen werden. Prinzipiell ähneln sich die Einträge in `terminfo` und `termcap`.

Wenn das Terminal durcheinander ist

Wie oben gezeigt wurde, werden Terminals durch die Ausgabe von Kontrollsequenzen gesteuert. Erhält das Terminal falsche Sequenzen, kann es passieren, dass man keine Ausgaben mehr erhält. Dies kann beispielsweise dadurch passieren, dass man versehentlich Binärdateien mit `cat` anzeigen lässt. Besonders unangenehm ist dies, wenn es bei den eingebauten Konsolen passiert, da man diese nicht einfach ein- und ausschalten kann, ohne den gesamten Computer herunterzufahren. Hier hilft oft der Befehl:

```
stty sane
```

Vor und hinter dem Befehl sollte man `ctrl-J` statt der Return Taste benutzen, da das Terminal eventuell so durcheinander ist, dass es zusätzlich zum `ctrl-J` ein `ctrl-M` erzeugt, was die Shell nicht richtig interpretieren kann. UNIX verwendet allein `ctrl-J` als Zeilenvorschub, während andere Systeme Kombinationen aus `ctrl-M` und `ctrl-J` benutzen.

Der Befehl `stty` dient in erster Linie zur Einstellung von Terminaleigenschaften wie der Baudrate. Die Option `sane` ist allerdings wohl die wichtigste Anwendung des Befehls. Sie setzt alle Einstellungen des Terminals auf einen Standardwert.

« [terminfo](#) | **[Terminals](#)** | [Anschluss eines Modems](#) »

Anschluss eines Modems

Neben der Verwendung des Modems als Netzbrücke ins Internet wird ein Modem manchmal auch als Wartungszugang verwendet. Man meldet sich wie an einem Textterminal an, nur dass die Anbindung über eine Telefonleitung läuft. Dieser Zugang öffnet nicht den gesamten Netzwerkzugang, reicht aber völlig aus, um die üblichen Administrationstätigkeiten durchzuführen.

Ein solcher Anschluss eines Modems ist prinzipiell nicht sehr viel anders als der Anschluss eines Terminals. Allerdings wird durch `getty`, der eventuell noch in der `inittab` bzw. der `tty` steht, der serielle Port für andere Anwendungen, die das Modem verwenden wollen, gesperrt. Diese müssen entfernt werden. Oft findet man auch die Modemkonfiguration in `inittab` oder `tty` fertig konfiguriert und lediglich auskommentiert.

Für die Verwendung eines Modems wird oft ein anderer Eintrag im Verzeichnis `/dev` verwendet. `getty` wird durch ein Flag an die etwas andere Benutzung angepasst, oder es gibt sogar ein alternatives `getty` wie `uugetty` unter HP-UX oder `mgetty` unter Linux.

Um das Modem mit unterschiedlichen Baudraten von außen ansprechen zu können, werden die Baudraten mit Kommata getrennt hinter den Aufruf von `getty` geschrieben. Auf diese Weise probiert das System die verschiedenen Baudraten der Reihe nach durch. Das Folgende ist ein Beispiel unter SCO:

```
getty -h ttyF01 19200, 9600, 4800, 2400
```

Unter LINUX ist gibt es für den Anschluss eines Modems das Programm `mgetty`. Der vollständige Eintrag der `/etc/inittab` lautet: vgl. Endres, Schmidt: Bei Anruf Netz. c't 25/99, S. 218-223.

```
m:23:respawn:/usr/sbin/mgetty -s 115200 /dev/ttyS1
```

`mgetty` erhält als Parameter die Geschwindigkeit der Rechnerschnittstelle und nicht die des Modems. Das Device `ttyS1` stellt die zweite serielle Schnittstelle dar. Ist nur ein Modem angeschlossen, ist ein symbolischer Link namens `modem` auf die entsprechende Schnittstelle sinnvoll:

```
ln -s /dev/ttyS1 /dev/modem
```

Tuning

Jeder Computerbenutzer träumt davon, mit ein paar gekonnten Eingriffen in das System dem Rechner Flügel zu verleihen. Jede Woche findet man solche Tipps in großer Vielfalt in den Computerzeitschriften, und nur der nachdenkliche Anwender stellt sich vielleicht hin und wieder die Frage, warum den Herstellern nicht in den Sinn gekommen ist, ihr System auf diese Art zu beschleunigen.

Tatsächlich gibt es einige Tricks, die ein etwas flotter laufendes System bewirken. Aus einer überlasteten Maschine machen diese Maßnahmen aber noch keinen Raser. Im Normalfall helfen Eingriffe in die Konfiguration, die ursprüngliche Geschwindigkeit zu erhalten oder die Last so zu verteilen, dass kein Engpass entsteht.

Irgendwann muss man mit Hardware aufrüsten. Bevor man aber einkauft, sollte man das System beobachtet haben, damit man auch sicher weiß, wo der Flaschenhals sitzt. So kann beispielsweise der beinahe immer richtige Tipp, den Speicher aufzurüsten, unsinnig sein, wenn auf der Maschine so viel Speicher vorhanden ist, dass kein Swapping feststellbar ist (siehe S. swap).

Unterabschnitte

- [Optimierung des Dateisystems](#)
 - [Überfüllung der Dateisysteme vermeiden](#)
 - [Blockgröße](#)
 - [Verteilung auf mehrere Platten](#)
 - [Eigenes Dateisystem für /tmp](#)
 - [Übervolle Verzeichnisse entsorgen](#)
- [Wissen, wo der Schuh drückt](#)
 - [vmstat](#)
 - [sar](#)
 - [Gegenmaßnahmen](#)

Unterabschnitte

- Überfüllung der Dateisysteme vermeiden
 - Blockgröße
 - Verteilung auf mehrere Platten
 - Eigenes Dateisystem für /tmp
 - Übervolle Verzeichnisse entsorgen
-

Optimierung des Dateisystems

Der Einfluss des Dateisystems auf den Gesamtdurchsatz ist natürlich davon abhängig, für welchen Einsatz die Maschine gedacht ist. Eine CAD-Workstation und auch eine Entwicklungsmaschine für einen Softwareentwickler werden von der Beschleunigung eines Dateisystems nicht profitieren. Dagegen werden alle Programme, die in größerem Umfang mit Daten arbeiten, dadurch zentral beeinflusst.

Überfüllung der Dateisysteme vermeiden

Ist die Platte über 90% gefüllt, wird das die Performance beeinflussen. Da nur wenig Platz ist, werden alle Lücken aufgefüllt, die sich durch das Löschen von Dateien im Laufe der Zeit gebildet haben. Nach einiger Zeit sind diejenigen Dateien, die regelmäßig geändert werden, die Lückenfüller der Platte und verursachen heftige Bewegungen des Schreib-/Lesekopfs. Man spricht in diesem Falle von Fragmentierung oder Zerclusterung eines Dateisystems.

Auf einer zentralen Servermaschine wird man die Belegung der Dateisysteme, auf denen Daten bewegt werden, möglichst unter 80%, besser unter 70% halten.

Defragmentierungssoftware, wie man sie vom PC her kennt, wird der eine oder andere unter UNIX vielleicht vermissen. Der Grund dafür ist, dass man sie unter UNIX normalerweise nicht braucht. Gibt es tatsächlich Grund zu der Annahme, dass die Platte durch Fragmentierung zu langsam ist, wird man über kurz oder lang eine größere Platte kaufen und die Daten beispielsweise per tar auf die neue Platte kopieren. Bei diesem Neuaufspielen der Daten werden alle Fragmentierungen beseitigt. Alternativ können Sie natürlich auch eine Bandsicherung durchführen, mit mkfs oder newfs ein neues Dateisystem erstellen und die Bandsicherung zurückholen.

Blockgröße

Auf älteren Systemen kann man die Blockgröße noch als Parameter beim Erzeugen des Dateisystems mit mkfs festlegen. Typischerweise liegt sie zwischen 512 Byte und 4 KByte. Je größer der Block ist, den das System mit einem Mal liest, desto geringer wird der Einfluss der langsamen Plattenzugriffe auf die gesamte Dateioperation. Das Lesen eines größeren Blocks kostet nur geringfügig mehr Zeit, da der Lesekopf schon an der richtigen Stelle ist. Beim nächsten Dateizugriff befindet sich dann der angeforderte Bereich schon im Puffer und man kann den Zugriff auf die Platte komplett sparen. Ab einer gewissen Blockgröße allerdings kippt dieser Wert wieder, wenn zu oft mehr geladen wird, als tatsächlich benötigt wird.

Der Platzbedarf der kleinsten Datei entspricht der Größe eines Blocks, da die Platte immer blockweise belegt wird. Bei vielen kleinen Dateien wird der verschwendete Speicherraum entsprechend groß.

Moderne Dateisysteme sind in der Lage, durch dynamische Caches und Aufspaltung von Blöcken für kleine Dateien diese Tuningmaßnahmen selbst zu übernehmen.

Verteilung auf mehrere Platten

Besitzt man mehrere physische Platten, kann man durch geschicktes Verteilen der Dateien einen Performancegewinn erzielen. Wenn zwei Dateien auf einer Platte liegen, auf die ständig wechselnd zugegriffen wird, muss der Schreib-/Lesekopf des Laufwerkes ständig zwischen diesen beiden Dateien hin- und herpositioniert werden. Kann man die beiden Dateien auf zwei Platten verteilen, werden diese Positionierungen eingespart. Gerade unter UNIX ist das Verteilen auf mehrere Platten extrem einfach. Durch einen symbolischen Link merken die zugreifenden Programme nicht einmal, dass eine Datei nicht mehr an der ursprünglichen Stelle liegt. Der zusätzliche Aufwand durch den Link ist minimal und wird auch nur einmal beim Öffnen benötigt. Danach arbeitet das Programm mit einem Dateihandle direkt auf der Datei.

Es versteht sich eigentlich von selbst, dass ein Verteilen der Dateien auf mehrere Partitionen der gleichen Platte kontraproduktiv ist und das Laufzeitverhalten verschlechtert.

Eigenes Dateisystem für /tmp

Das Verzeichnis /tmp kann auf eine eigene Platte gelegt werden. Dies bringt auf Systemen etwas, die das Verzeichnis intensiv nutzen, wie beispielsweise bei der Kompilierung. Es wird eine höhere Geschwindigkeit erreicht, da das ständige Schreiben und Löschen zu einer starken Zerclusterung führt. Da /tmp aber jederzeit gelöscht werden kann, ist es möglich, durch rekursives Löschen des kompletten Verzeichnisses mit einem zusammenhängenden System weiterzuarbeiten. Da im Bereich des /tmp-Verzeichnisses bei einem Absturz häufig ein unzusammenhängendes Dateisystem entsteht, ist es von Vorteil, wenn das komplette Dateisystem /tmp bedenkenlos gelöscht werden kann.

Ist ein eigenes Dateisystem für /tmp nicht praktikabel, sollte man von Zeit zu Zeit im Single-User-Modus das Verzeichnis komplett entfernen und wieder neu anlegen, da so auch das Verzeichnis wieder geleert wird. Insbesondere wenn die Größe des Verzeichniseintrags in der Ausgabe von `ls -ld` auffallend groß ist, sollte man diese Maßnahme ergreifen.

Bei den eben erwähnten Entwicklermaschinen, bei denen der Compiler oft das Verzeichnis /tmp benutzt, wird manchmal zur Beschleunigung das Verzeichnis in eine RAM-Disk gelegt. Natürlich macht das nur Sinn, wenn die Maschine üppig mit Speicher ausgestattet ist.

Übervolle Verzeichnisse entsorgen

Verzeichnisse sind lineare Strukturen. Wird eine Datei angelegt, wird sie hinten im Verzeichnis angelegt. Die Einträge sind also nicht alphabetisch geordnet, wie es bei der Anzeige erscheint. Also wird bei der Suche nach einer Datei die Liste von vorn nach hinten durchgegangen. Da die Suche nach dem Dateinamen nur relevant wird, wenn die Datei immer wieder geöffnet und geschlossen wird, fallen Probleme in diesem Bereich nicht so sehr auf. Wenn Verzeichnisse aber sehr voll werden, werden die Zugriffe immer langsamer. Besonders kritisch wird es, wenn in Verzeichnissen regelmäßig gelesen und

geschrieben wird.

Ein Merkmal dafür, dass ein Verzeichnis überlastet ist, ist seine Größe. Man kann dies leicht durch den Befehl `ls -ld` prüfen. Im Allgemeinen werden die meisten Verzeichnisse gleich groß sein. Bei den Verzeichnissen `/dev` und `/tmp` dürfte man bereits sehen, dass sich darin deutlich mehr Dateien befinden. Verzeichnisse werden aber normalerweise nicht von selbst wieder kleiner. Um ein Verzeichnis, in dem viel geschrieben und gelöscht wurde, wieder auf eine normale Größe zu bringen, sollte man zunächst eine Sicherung der Daten per `tar` (siehe S. `tar`) durchführen. Anschließend löscht man per `rm -r` den gesamten Ast inklusive Verzeichnis und holt dann die gesicherten Dateien wieder an den Ort zurück. Dabei wird das Verzeichnis neu angelegt.

« [Tuning](#) | [Tuning](#) | [Wissen, wo der Schuh](#) »

Unterabschnitte

- [vmstat](#)
- [sar](#)
- [Gegenmaßnahmen](#)

Wissen, wo der Schuh drückt

Die Einschätzung, dass die Maschine langsam ist, reicht nicht aus, um Gegenmaßnahmen zu ergreifen. Man muss schließlich die Beschränkung beseitigen, die die Leistung am meisten einschnürt.

vmstat

`vmstat` ist ein Programm, das auf fast jeder UNIX-Maschine verfügbar ist. Es wertet diverse Kernelprotokolle aus und stellt sie dar. Dabei werden die Prozesse, der Speicher, das Swapping, der Plattendurchsatz und die CPU-Belastung beobachtet. Der Aufruf lautet:

vmstat Sekundenabstand Wiederholungen

Der erste Parameter bestimmt, wie viel Zeit in Sekunden zwischen den Ausgaben vergehen soll. Aufgrund der vielen Daten, die erhoben werden, sollte der Abstand nicht allzu gering sein, damit `vmstat` nicht selbst die Messung verfälscht. Der zweite Parameter bestimmt die Anzahl der Messungen. Multipliziert man die beiden Werte, erhält man den Zeitraum, den die Messung abdeckt. `vmstat` erzeugt eine Ausgabe, die wie folgt aussieht:

procs			memory				swap		io		system		cpu		
r	b	w	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id
1	0	0	512	12932	90984	116120	0	0	0	0	103	407	5	0	95
1	0	0	512	12932	90984	116120	0	0	0	0	123	502	4	0	96

Folgende Werte werden dabei gemessen:

- [procs:] zählt r (warten auf Laufzeit), b (uninterruptable sleeping), w (swapped processes)
- [memory:] aus dem swpd (virtueller Speicher), free (idle), buff und cache
- [swap:] swap in (si), swap out (so)
- [io:] blocks in (bi), blocks out (bo)
- [system:] in (Interrupts per second), cs (context switch per second)
- [cpu:] Verteilung der CPU-Last auf user (us), system (sy) und idle (id). Beispielsweise deutet ein starkes Swapping bei gleichzeitiger hoher CPU-Belastung im Systembereich auf zu wenig Hauptspeicher hin.

Besonders interessant werden diese Werte, wenn man sie zu verschiedenen Zeitpunkten des Tages aufnimmt. Dadurch kann man erkennen, welche Zahlen sich überdurchschnittlich stark verändern. Es

ist wichtig zu wissen, welche Werte für die Maschine im Ruhezustand typisch sind. Welche Ergebnisse erhält man bei einer beschäftigten Maschine, die aber noch zügig reagiert, und wie sehen die Zahlen aus, wenn die Maschine überlastet ist? Ein Gefühl für diese Zahlen sollten Sie als Administrator möglichst schon haben, bevor die Geschäftsleitung nach dem Verantwortlichen ruft.

Im folgenden Abschnitt über `sar` wird gezeigt, wie man ein solches Analysetool in die `crontab` (siehe S. `crontab`) stellt. Ähnlich kann man auch mit `vmstat` arbeiten.

sar

Ein anderes noch umfangreicheres Beobachtungstool gibt es bei System V Maschinen. Es heißt `sar` (system activity report). Es ist normalerweise nicht aktiviert. Man muss es also in Betrieb nehmen, um Daten über das System zu sammeln. `sar` erfasst Aktivitätszähler, die das System führt. Vor allem erstellt `sar` regelmäßige Statistiken, die die Auswertung eines Vergleichs zwischen Last- und Ruhezeiten ermöglichen. Da `sar` recht gute Informationen liefert, wenn etwas schief läuft, sollte man es in jedem Fall starten, wenn eine neue Maschine in Betrieb genommen wird, wenn man Probleme hat, deren Herkunft unerklärlich sind, oder wenn sich Dinge grundsätzlich ändern, wie der Einsatz neuer Software oder Hardware.

Die gesammelten Daten werden in dem Pfad `/var/adm/sa` gesammelt. Eventuell muss das Verzeichnis erst angelegt werden. Darin legt `sar` für jeden Tag des Monats eine Datei mit dem Präfix `sa` an. Für den 12. des Monats wäre dies also `sa12`.

Damit diese Dateien entstehen, ist ein Datensammler erforderlich. Dieser heißt `sadc` und wird in der `rc`-Datei des Systems gestartet:

```
/usr/sbin/sa/sadc /var/adm/sa/sa`date +%d`
```

Darüber hinaus verbirgt er sich im Skript `sa1`, das normalerweise in die `crontab` (siehe S. `crontab`) des `root` eingetragen wird. vgl. Manpage zu `sa1`

```
0 * * * 6,0 /usr/lib/sa/sa1 3600 /var/adm/sa/sa`date +%d`
0 8-17 * * 1-5 /usr/lib/sa/sa1 3600 /var/adm/sa/sa`date +%d`
0 8-17 * * 1-5 /usr/lib/sa/sa1 1200 3 /var/adm/sa/sa`date +%d`
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A
```

Das Skript `sa2` erzeugt aus den `sa`-Dateien einen täglichen Bericht in reinem Textformat, den es unter fast gleichem Namen anlegt. Das Präfix ist allerdings hier `sar` statt `sa`.

Zu Auswertung der `sa`-Dateien wird das Kommando `sar` verwendet. Durch seine Optionen kann bestimmt werden, welche Informationen angezeigt werden:

- [-A] Alle Optionen auf einmal einschalten.
- [-a] Dateizugriffe. Zeigt `iget/s`, `namei/s` und `dirblk/s`
- [-B] Kopierpufferaktivitäten
- [-b] Pufferaktivitäten:

`bread/s`, `bwrite/s` sind die Blockzugriffe zwischen den Systempuffern und der Peripherie.

lread/s, lwrite/s sind die Zugriffe auf die Systempuffer.

%rcache, %wcache ist das Verhältnis zwischen Zugriffen auf die Puffer und die Peripherie.

- [-c] Berichtet über Systemcalls, unter anderem über fork() und exec().
- [-d] Zeigt die Aktivität einzelner Blockdevices. Diese Informationen können hilfreich sein, um Zugriffe auf mehrere Platten zu verteilen.
- [-g] Serielle Schnittstellen
- [-h] Pufferstatistiken
- [-m] Messages und Semaphore
- [-n] Namenscache-Statistik
- [-p] Paging-Aktivitäten. Diese Statistik gibt Auskunft über die Häufigkeit des Ladens ausgelagerter Speicherbereiche.
- [-q] Gibt Auskunft über die Prozesslisten.
- [-R] Prozessaktivitäten: beispielsweise wie oft der Prozessumschalter gestartet wurde.
- [-u] CPU-Auslastung: %usr, %sys, %wio, %idle

usr ist die Auslastung durch Anwendungsprogramme, sys ist der Verbrauch durch Systemfunktionen. wio ist das Warten auf I/O-Funktionen. Unter idle steht die Zeit, die sich die CPU gelangweilt hat. Diese sollte nur in Ausnahmefällen unter 50% fallen. Insbesondere, wenn dieser Wert über 90% steigt, besteht der Verdacht, dass ein Programm Polling betreibt oder dass das System ein Problem hat.

- [-v] Status der Prozess-, i-node- und Dateitabellen.
- [-w] Swapping und Switching: pswch/s zeigt die Anzahl der Prozesswechsel pro Sekunde.
- [-y] TTY-Devices

Gegenmaßnahmen

Die CPU-Last muss nach User- und nach Systemlast getrennt beurteilt werden. Beide haben unterschiedliche Ursachen. Die Systemlast sagt, dass die Maschine lange mit Systemaufrufen wie dem Laden von Dateien, der Netzkommunikation oder Ähnlichem zu tun hat. Dabei tragen beispielsweise beim Laden einer Datei nicht so sehr die Plattengeschwindigkeit, sondern der Aufwand beim Verwalten der Zugriffe in die CPU-Systemlast bei. Dazu gehört beispielsweise das Prüfen von Sperren oder Quota. Eine hohe Systemlast kann bedeuten, dass zu wenig RAM in der Maschine vorhanden ist und dass das System ständig mit dem Swappen befasst ist. Es kann aber auch bedeuten, dass zu wenig Puffer für die Dateioperationen zur Verfügung steht.

CPU-Last im Userbereich bedeutet, dass Programme sich stärker mit ihrem Code als mit den Daten beschäftigen. Das geht in Ordnung, wenn es Programme sind, die beispielsweise dreidimensionale Bilder berechnen. Auch bei der Kompilierung auf Entwicklermaschinen können kurzfristig bis zu 100% CPU-Auslastung im Userbereich entstehen. Bei den üblichen Verwaltungsprogrammen auf Servern sind CPU-Belastungen von 10% bis 20% bereits relativ viel. Wenn der Wert darüber hinausgeht, muss man feststellen, welches Programm wie viel CPU-Zeit beansprucht. Zu diesem Zweck gibt es Programme wie top (siehe S. top), das quasi eine Hitparade der Prozesse darstellt. Verfügt man nicht über ein derartiges Programm, kann man auch durch mehrfaches Anzeigen der Prozessliste mit ps die schuldigen Prozesse finden. Da ps immer nur die insgesamt angefallene CPU-Zeit anzeigt, muss man die Differenz der Messungen bilden. Ein Programm, das auffallend viel Zeit verbraucht, kann entweder völlig durcheinander sein oder durch so genanntes Polling (siehe S. polling) CPU-Zeit verschwenden.

Im I/O-Bereich zeigt sich die Auslastung der Dateisysteme. Interessant ist die Größe des Cache. Ungepufferte Plattenzugriffe sind extrem langsam. Wenn die Puffergröße nicht ausreicht, werden spürbare Performanceeinbrüche die Folge sein. In einigen älteren Systemen (beispielsweise SCO 3.x) muss der Plattencache statisch als Kernelparameter festgelegt werden. Für einen Firmenserver ist der Defaultwert oft viel zu klein und muss unbedingt heraufgesetzt werden. Bei Systemen mit dynamischer

Speicherverteilung wird der Cache natürlich nicht erhöht, wenn dadurch der Hauptspeicher so knapp wird, dass das Swapping erheblich zunimmt. In solch einem Fall ist ein Speicherausbau dringend geboten.

Dass der freie Hauptspeicher einer Maschine mit dynamischer Speicherverwaltung gering ist, sollte niemanden alarmieren. Aus der Sicht des Speichermanagers ist freier Speicher nur ein unnützer Stromfresser, und er wird versuchen, die wertvolle Ressource auf die Plattenpuffer zu verteilen. Dort macht er sich nützlich, indem er Dateizugriffe beschleunigt. Auch ist es völlig normal, dass erhebliche Teile des Swapbereichs belegt sind, obwohl freier Speicher zur Verfügung steht. Sofern die ausgelagerten Prozesse nicht aktiv werden, können sie getrost dort bleiben, wo sie sind.

« [Optimierung des Dateisystems](#) | [Tuning](#) | [Informationen sammeln](#) »

Informationen sammeln

Auch wenn UNIX-Maschinen normalerweise extrem zuverlässig sind, gibt es Stromausfälle, Programmierfehler, Anwenderfehler, Hardwareausfälle oder die gefürchteten Sonnenwinde und UFOs. Beides sind beliebte Administratorenausreden, wenn die Ursache einer Panne nur mit großem Aufwand feststellbar wäre., die einem Administrator das Leben schwer machen. In diesem Fall ist es wichtig, die Ursache der Zwangspause möglichst schnell zu ermitteln. Da während dieser Suche normalerweise alle anderen Mitarbeiter nichts tun, außer die Gehaltskasse der Firma zu belasten, sollte man schnell und rational handeln.

Der erste Blick, noch bevor man sich einloggt, sollte der Konsole gelten, sofern man eine solche hat. Hier stehen gegebenenfalls Meldungen, die weiterscrollen könnten und die, aus welchen Gründen auch immer, vielleicht nicht in die Protokolldateien einfließen. Stellt man tatsächlich fest, dass Informationen auf der Konsole nicht in die Protokolldateien einfließen, sollte man die Konfiguration des syslog-Dämons genauer betrachten.

Unterabschnitte

- [Versionsinformationen: uname](#)
- [Der syslog-Dämon und die messages-Datei](#)
 - [Konfiguration von syslog.conf](#)
- [Umgang mit großen Protokolldateien](#)
- [Briefe aus dem Nirvana](#)
- [Bootzeitpunkt und Systemlast: uptime](#)
- [Prozessbeobachter](#)
 - [ps](#)
 - [Prozess-Hitparade: top](#)
- [Nicht immer mit Tötungsabsicht: kill](#)
 - [SIGHUP](#)
 - [Priorität ändern statt töten](#)
 - [Kurzfristiges Parken](#)
 - [Terminieren](#)
- [Offene Dateien](#)
- [Programmzusammenbrüche \(Core Dump\)](#)
- [Systemabsturz \(Kernel-Panic\)](#)

Versionsinformationen: uname

Wenn man die Maschine nicht kennt, sollte man sich erst informieren, was für ein System man vor sich hat. Die Maschine, das Betriebssystem und die Version werden von dem Kommando `uname` ausgegeben.

```
gaston# uname -a
Linux gaston 2.2.16 #1 wed Aug 2 20:22:26 GMT 2000 i686 unknown
hpsrv$ uname -a
HP-UX hpsrv B.10.10 A 9000/816 1962328252 two-user license
note
$uname -a
FreeBSD note.willemer.edu 4.4-RELEASE FreeBSD 4.4-RELEASE #0:
Tue Sep 18 11:57:08 PDT 2001      murray@builder.FreeBSD.org:/usr
/src/sys/compile/GENERIC i386$
[tex2html\_wrap\_inline5510]
```

[Optionen von `uname`]C|L|L|L|L Flag & Bedeutung & Beispiel Linux & Beispiel HP & Beispiel FreeBSD
-s & Name des Betriebssystems & Linux & HP-UX & FreeBSD
-n & nodename & gaston & hpsrv & note.willemer.edu
-r & release & 2.2.16 & B.10.10 & 4.4-RELEASE
-m & machine & i686 & 9000/816 & i386

Die Informationen von `uname` sollte man auch greifbar haben, wenn man eine Hotline kontaktiert. Die Wahrscheinlichkeit ist hoch, dass man nach diesen Informationen gefragt wird.

Unterabschnitte

- [Konfiguration von syslog.conf](#)

Der syslog-Dämon und die messages-Datei

Tritt ein Problem mit der Maschine auf, sollte man sich zuerst die Protokolldateien des Dämons `syslogd` ansehen. Man findet sie normalerweise in den Verzeichnissen `/var/log` oder `/var/adm`. Über den `syslog`-Dämon setzen die meisten sauber geschriebenen Programme ihre Fehlermeldungen ab. Insbesondere die Datei namens `messages` oder `syslog` ist interessant, da dies normalerweise die zentrale Datei des `syslog`-Dämons ist.

Welche Dateien zur Protokollierung verwendet werden, wird in der Datei `/etc/syslog.conf` protokolliert, die unten näher beschrieben ist.

In der Datei `/var/log/messages` finden sich alle Ereignisse historisch sortiert. Alle sind mit Datum und Uhrzeit versehen. Man braucht ein wenig Übung, um sich in dieser Datei zurechtzufinden. Darum ist es durchaus sinnvoll, sie auch dann zu lesen, wenn gerade mal nichts passiert ist. Ungewöhnliches kann man dann im Ernstfall schneller identifizieren.

Konfiguration von syslog.conf

Der `syslogd` liest beim Start oder bei jedem `SIGHUP`-Signal die Datei `/etc/syslog.conf` und entnimmt ihr, welche Meldungen gewünscht werden und wo sie abzulegen sind. Die Zeilen in der `syslog.conf`-Datei beginnen mit der Facility, die die Ursache des Fehlers beschreibt. Durch einen Punkt getrennt folgt der Level, also die Heftigkeit des Fehlers. Danach muss zwingend ein Tabulatorzeichen stehen, und dann folgt der Ausgabekanal, in den die Fehlermeldungen geschrieben werden. Beispiel:

<code>news.crit</code>	<code>/var/log/news</code>
------------------------	----------------------------

Das bedeutet, dass die Fehler, die vom Newssystem erzeugt werden und die mindestens kritisch sind, in die Datei `/var/log/news` geschrieben werden sollen.

Die Facility, also der Verursacher, kann folgende Werte annehmen:

[Fehlerquellen]L|L Facility & Verursacher
kern & Kernel
user & Anwenderprogramme
mail & sendmail und Kollegen
daemon & Systemhintergrundprozesse
auth & Authorisierungsbefehle
lpr & Das Drucksystem
news & Der Newsserver
uucp & UUCP

cron & Der cron-Dämon
mark & Zeitstempel
local0 - local7 & frei verfügbar
* & alle

Mit dem Level kann festgelegt werden, wie gravierend das Problem sein muss, damit es in der Protokolldatei erscheint. Für jede Facility kann ein eigener Level bestimmt werden, ab dem eine Meldung erfolgt. Tabelle zeigt die Level mit nach unten abnehmender Bedeutung.

[Schwere des Fehlers]L|L Level & Bedeutung
emerg & Panik
alert & Alarm
crit & kritisch
warning & Warnmeldungen
notice & Hinweise
info & informell
debug & Ablaufnotizen von Programmen in der Entwicklung

Normalerweise wird als Ausgabe eine Datei oder auch das Konsolendevice angegeben. Die entsprechenden Dateien müssen nach dem Eintrag angelegt werden. Der syslog-Dämon kann nämlich keine Dateien erzeugen. Es können auch mehrere Ziele angegeben werden, die durch Kommata getrennt werden. Die Dateien müssen mit vollem Pfadnamen angegeben werden. Steht vor dem Ziel ein @, kann eine IP-Nummer oder ein Hostname angegeben werden. Dann wird die Fehlermeldung an den syslog-Dämon des betreffenden Rechners versandt.

Zur Veranschaulichung hier eine sehr kleine syslog.conf-Datei:

```
# /etc/syslog.conf - Configuration file for syslogd(8)
*.emerg                                *
mail.*                                /var/log/mail
*.info                                /var/log/messages
```

Der Stern als Ziel, wie oben bei den Emergency-Nachrichten bedeutet, dass die Mitteilung sofort an alle angemeldeten Terminals versandt wird. In der nächsten Zeile werden alle Nachrichten, die Mails betreffen, in die Datei mail eingetragen. Alle restlichen Meldungen, die mindestens den Level info haben, werden in die Datei messages geschrieben, also auch die emerg-Meldungen, die damit an zwei Ziele verteilt werden.

Unter Linux ist der ursprüngliche Befehlssatz für die Datei syslog.conf erweitert worden. Dadurch ist es möglich, auch über eine Pipe an ein Programm zu schreiben, indem ein Pipesymbol vor das Ziel gestellt wird. Der Level kann durch Voranstellen eines Gleichheitszeichens auf exakt diesen Level beschränkt werden. So ist es möglich, sich ausschließlich die info-Level-Meldungen in eine Datei ausgeben zu lassen, ohne alle höheren Level auch dort zu finden. Details und weitere Optionen findet man in den Manpages der syslog.conf.

Umgang mit großen Protokolldateien

Häufig ist es erforderlich, Protokolldateien ständig zu beobachten. Der Befehl `tail` zeigt immer die letzten Zeilen einer Datei an. Mit der Option `-f` wird auf dem Terminal jede neue Zeile angezeigt, die an diese Datei angehängt wird. Das Terminal ist blockiert, bis man die Beobachtung mit `ctrl-C` abbricht.

```
gaston# cd /var/log
gaston# tail -f messages
Jun 18 20:11:36 gaston pppd[2592]: sent [LCP TermReq id=0x4 "User request"]
Jun 18 20:11:36 gaston pppd[2592]: rcvd [LCP TermAck id=0x4]
Jun 18 20:11:36 gaston pppd[2592]: Connection terminated.
Jun 18 20:11:36 gaston pppd[2592]: Connect time 0.4 minutes.
Jun 18 20:11:36 gaston pppd[2592]: Sent 3333 bytes, received 25062 bytes.
Jun 18 20:11:36 gaston pppd[2592]: waiting for 1 child processes...
Jun 18 20:11:36 gaston pppd[2592]:  script /etc/ppp/ip-down, pid 2616
Jun 18 20:11:36 gaston pppd[2592]: script /etc/ppp/ip-down finished (pid 2616), status = 0x0
Jun 18 20:11:36 gaston pppd[2592]: Exit.
Jun 18 20:26:54 gaston su: (to root) arnold on /dev/pts/6
```

Wenn eine Protokolldatei lange Zeit nur gefüllt wurde, kann sie leicht einige Megabyte groß werden. Dann wird es schwierig, sie zu handhaben, um nach bestimmten Stichwörtern oder Tagen zu suchen. Typischerweise ist man an den etwas neueren Informationen interessiert. Mit `tail` kann man die letzten Zeilen herausholen und in eine andere Datei umleiten, die leichter zu bearbeiten ist. Mit einem Minuszeichen gefolgt von der Zahl, lässt sich angeben, wie viele Zeilen herauszuziehen sind. Beispiel:

```
tail -2000 messages > guckmal
```

Damit sind die letzten 2000 Zeilen in der Datei `guckmal` gelandet. Mit dieser Dateigröße lässt sich leicht arbeiten.

Es ist eine ganz schlechte Idee, eine Protokolldatei einfach zu löschen, wenn man der Meinung ist, dass sie zu voll geworden ist. Normalerweise ist diese Datei von dem Erzeuger des Protokolls geöffnet worden. Verschwindet die Datei, schreibt der Prozess ins Nirwana. Kaum ein Programm reagiert auf Fehler beim Schreiben, indem es die Datei neu anlegt. Um die Datei `/var/log/messages` auf 0 zurückzusetzen, geben Sie den folgenden Befehl ein:

```
> /var/log/messages
```

Diese etwas brutale Methode kann man auf seiner eigenen Workstation durchaus praktizieren. Auf einer Produktionsmaschine wird man Protokolle normalerweise eine Zeit lang archivieren. Die spontane Idee ist, die Datei `messages` zu kopieren und sie dann wie oben gesehen zu stutzen. Das Problem ist, dass dabei ein Zeitloch entsteht, in dem Fehlermeldungen verschwinden. Eine Fehlermeldung, die nach dem Erstellen der Kopie und vor dem Stutzen der `messages` entsteht, würde verloren gehen. Darum geht man so vor, dass man zunächst die Datei umbenennt. Der Dateizugriff

wird dadurch nicht gestört, da die Prozesse beim Bearbeiten von Dateien nur einmal beim Öffnen auf den Dateinamen verweisen. Danach wird nur noch über eine Kennung auf die Datei zugegriffen. Was währenddessen in den Verzeichniseinträgen passiert, ist dem Prozess ziemlich egal. Nun legt man eine neue Protokolldatei mit touch an. Die erzeugte Datei ist 0 Byte lang. Man sollte auch darauf achten, dass Benutzer und Rechte mit der Originaldatei übereinstimmen. Anschließend sendet man dem syslogd das Signal SIGHUP zu. Das bewirkt, dass er die syslog.conf-Datei noch einmal einliest und die dort aufgeführten Dateien neu eröffnet. Ab diesem Moment erfolgt die Protokollierung in der soeben angelegten Datei. Mit dem Befehl fuser kann man noch einmal kontrollieren, dass die Protokolldatei von keinem Prozess mehr bearbeitet wird. Danach kann man sie packen, damit sie nicht mehr so viel Platz wegnimmt. Am Beispiel der Datei messages sähe der Ablauf so aus:

```
gaston# cd /var/log
gaston# fuser messages
messages:                 378
```

Prozess 378 greift auf die Datei messages zu. Vermutlich ist das der syslogd.

```
gaston# mv messages messages.$(date +%y%m%d)
gaston# fuser messages.
$(date +%y%m%d)
messages.020224:          378$
```

Die Datei wurde nun umbenannt. Immer noch greift der Prozess mit der PID 378 auf sie zu, obwohl der Name der Datei sich geändert hat.

```
gaston# touch messages
gaston:/var/log # l messages*
-rw-r--r-  1 root  root          0 Feb 24 12:07 messages
-rw-r---  1 root  root 2447648 Feb 24 12:06 messages.020224
gaston# chmod 640 messages
gaston# l messages*
-rw-r---  1 root  root          0 Feb 24 12:07 messages
-rw-r---  1 root  root 2447648 Feb 24 12:06 messages.020224
```

Die neue Protokolldatei wurde erfolgreich angelegt und mit den Rechten der alten versehen. Auch der Eigentümer stimmt überein.

```
gaston# ps -ef | grep syslog
root      378      1  0 10:24 ?        00:00:00 /sbin/syslogd
root     1486   1465  0 12:09 pts/2    00:00:00 grep syslog
gaston# kill -SIGHUP 378
```

Sicherheitshalber wird noch einmal geprüft, ob der Prozess 378 tatsächlich der syslogd ist. PID 1486 ist der eigene Prozess, der die Prozessliste durchsucht. Nun wird mit dem Befehl kill der syslogd aufgefordert, die Konfiguration neu zu lesen und damit alle Dateien neu zu eröffnen.

```
gaston# fuser messages
messages:                 378
gaston# gzip messages.$(date +%y%m%d)
```

Erwartungsgemäß hat der syslogd die Datei messages und damit die gerade neu untergeschobene eröffnet. Die alte Protokolldatei kann gepackt werden.

Mit Hilfe des Befehls `date` wird der neue Dateiname mit dem aktuellen Datum gekennzeichnet. Die Konstruktion `$(Kommando)` bewirkt dasselbe wie die so genannten Backquotes ``Kommando``, lässt sich aber etwas leichter lesen. Das Ergebnis des eingeklammerten Kommandos wird ermittelt und in den Befehl integriert.

Da solche Arbeiten auf Produktionsmaschinen regelmäßig auftreten, ist es ein nahe liegender Gedanke, sie in die `crontab` (siehe S. `crontab`) zu integrieren.

« [Der syslog-Dämon und die](#) | [Informationen sammeln](#) | [Briefe aus dem Nirvana](#) »

Briefe aus dem Nirvana

Einige Programme melden ihre Fehler nicht über den syslog-Dämon, sondern verbreiten ihre Klagen per Mail. Prominentester Vertreter dieser Art ist der Druckdämon, der fehlgeschlagene Druckaufträge an den Benutzer sendet, der den Auftrag gab. `crontab` und `at` senden ihre komplette Ausgabe an den Besitzer. Darum gehört ein mindestens lokal arbeitendes Mailsystem zur Standardinstallation. Dazu gehört auch das Standardtool `mail`. Leider ist die Benutzung etwas antiquiert. Es ist darum eine gute Idee, ein einfaches, aber übersichtliches Mailprogramm wie `elm` oder `pine` zu installieren. Ein grafisches Tool braucht recht viele Ressourcen, die bei einem Zusammenbruch des Servers vielleicht nicht zur Verfügung stehen.

Sollte ein Auslesen der Mail mit Hilfe der installierten Programme nicht möglich sein, kann man als `root` natürlich auch die E-Mail lesen, indem man mit dem Editor im Verzeichnis `/var/spool/mail` die Datei `root` durchsieht.

Man kann und sollte die Mail, die an `root` geht, weiterleiten (siehe S. `mailforward`). Steht der Rechner nicht unter besonderer Aufsicht, kann man sie sogar an einen fremden Rechner senden. Auf diese Weise merkt man schnell, dass etwas schief läuft. Dennoch sollte man ein Duplikat auf dem Rechner lassen, sodass die Mail an der Stelle vorhanden ist, wo das Problem ausgelöst wurde. Ein Beispiel für eine solche Konfiguration findet sich in Kapitel ab S. `mailforward`.

Bootzeitpunkt und Systemlast: uptime

Das Programm `uptime` ist ein kleines hilfreiches Programm, das man auf jeder UNIX-Maschine findet. Eine wichtige Information kann man durch den Aufruf des Befehls `uptime` erhalten. Er liefert eigentlich die Information, seit wann die Maschine ununterbrochen läuft. Das kann einen Hinweis darauf geben, ob die Maschine zwischendurch zusammengebrochen ist oder abgeschaltet wurde.

Als weiterer Wert wird die durchschnittliche Systemlast (load average) angezeigt. Schauen Sie sich diesen Wert hin und wieder an, damit Sie ein Gefühl dafür bekommen, wie sich die Maschine im Normalzustand verhält. Als Faustregel zeigt ein Wert über 1 eine gut ausgelastete Maschine. Ab etwa 3 steht die Maschine unter Last.

```
gaston> uptime
 5:11pm up 7:33,  1 user,  load average: 0.06, 0.01, 0.00
gaston> uptime
11:11pm up 14:40,  1 user,  load average: 0.95, 0.47, 0.28
```

Im Beispiel sind zwei unterschiedliche Aufrufe von `uptime` angezeigt. Die erste Meldung besagt, dass es 17:11 ist, dass der Rechner seit 7 Stunden und 33 Minuten läuft, dass ein Benutzer eingeloggt ist und dass es der Maschine richtig langweilig ist. Der zweite Aufruf wurde parallel zu einem Compilerlauf kurz vor dessen Ende ausgeführt. Hier ist es 23:11, die Maschine läuft seit 14 Stunden, 40 Minuten, hat ebenfalls einen Benutzer, und die Systemlast liegt bei 0.95. Andere Prozesse würden auf diesem System noch flüssig laufen, allerdings nicht ganz so schnell wie auf einer unbeschäftigten Maschine.

Unterabschnitte

- [ps](#)
- [Prozess-Hitparade: top](#)

Prozessbeobachter

Ein Prozess ist ein gestartetes Programm. Es hat damit seinen Ursprung in einer ausführbaren Datei. Der Prozess beansprucht Platz im Hauptspeicher für seinen Programmcode sowie seine Daten und erwartet die Zuteilung des Prozessors. Alle Prozesse haben einen Elternprozess. Bei einem von der Shell aus gestarteten Programm ist der Vater die Shell. Die Shell ist beispielsweise wiederum Kind eines Loginprozesses. Daraus ergibt sich ein Baum von Prozessen, der letztlich auf den `init`-Prozess zurückgeht, der beim Booten entsteht. `init` hat immer die Prozessnummer 1. Die dann entstehenden Prozesse bekommen bei ihrer Entstehung eine neue, derzeit freie Nummer und behalten sie, bis sie enden. Diese Prozessnummer wird Prozess-ID oder PID genannt.

Jeder Prozess hat einen eindeutigen Besitzer. Das ist derjenige, der den Prozess gestartet hat. Lediglich, wenn die aufgerufene Programmdatei das User-ID-Bit gesetzt hat (siehe S. `suid`), ergibt sich eine besondere Situation. Der Prozess hat nach wie vor den realen Benutzer, der den Prozess gestartet hat. Die effektiven Ausführungsberechtigungen erhält er aber durch den Besitzer des gestarteten Programms. Er läuft also quasi unter der Flagge des Besitzers der Programmdatei. Darum spricht man hier vom realen Benutzer und vom effektiven Benutzer.

Alle lauffähigen Prozesse werden in eine Prozessliste eingetragen. Das Betriebssystem startet einen Prozess aus dieser Liste für eine Weile, bis eine gewisse Zeit vergangen ist. Dann wird der Prozess eingefroren. Das heißt, man kopiert den Prozessorstatus in einen Speicher und holt den nächsten Prozess aus der Liste, kopiert den vorher im Speicher abgelegten Prozessorstatus wieder in den Prozessor und lässt diesen Prozess laufen, bis der nächste Takt erreicht ist. Da der Takt relativ schnell ist, bemerken die Anwender kaum, dass ihre Programme kurzzeitig inaktiv sind. Stattdessen entsteht der Eindruck, dass alle Prozesse parallel laufen. Der Teil des Betriebssystems, der diese Prozessumschaltung durchführt, heißt Scheduler.

Es gibt bestimmte Situationen, in denen ein Prozess nicht weiterarbeiten kann. Das kommt beispielsweise vor, wenn er auf einen Tastendruck des Benutzers wartet oder darauf, dass ein Datenblock von der Platte gelesen wird oder dass sonst ein bestimmtes Ereignis eintritt. Wenn der Prozess derartige Anfragen an das Betriebssystem stellt, wird er aus der normalen Prozessliste herausgenommen und in eine Warteschlange gesteckt. Sobald das erwartete Ereignis eintritt, holt das Betriebssystem den Prozess aus der Warteschlange und bringt ihn wieder in die Prozessliste.

Einer der Vorteile von UNIX ist die Möglichkeit, wirklich alle laufenden und wartenden Prozesse, ihre Zugehörigkeit und ihren Ressourcenverbrauch beobachten zu können. Als Administrator hat man dadurch viele Möglichkeiten zu erkennen und einzugreifen, wenn etwas nicht sauber läuft.

ps

Der Befehl `ps` zeigt eine Liste der aktuell laufenden Prozesse an. Der Befehl kennt sehr viele Optionen, die bestimmen, welche Informationen angezeigt werden. Leider unterscheiden sich die Optionen zwischen BSD-Systemen und System V erheblich. Linux versucht einen Mittelweg, indem es normalerweise die System V-Optionen verwendet, allerdings auch die BSD-Optionen kennt, wenn man den Bindestrich vor den Optionen weglässt. MacOS X verwendet die Syntax von BSD.

Für eine Übersicht über alle auf dem System laufenden Prozesse gibt es Kombinationen von Optionen, die der Administrator fast ohne nachzudenken verwendet.

[Varianten des `ps`][L]L Befehl & System

`ps -alx` & BSD und Linux: zeigt auch PID und PPID

`ps -aux` & BSD: zeigt auch Benutzer

`ps -elf` & System V, beispielsweise SCO und HP-UX

Ohne Parameter werden nur die Prozesse gezeigt, die von dieser Sitzung erzeugt wurden. In Tabelle sind die Optionen aufgeführt, die BSD-Systeme verwenden.

[Beliebte `ps`-Optionen (BSD)][C]L Option & Anzeige

`-a` & Alle Prozesse, nicht nur die eigenen

`-x` & Zeigt auch Prozesse, die keinem Terminal zugeordnet sind (z. B. Dämonen)

`-u` & Zeigt den Benutzer des Prozesses und die Startzeit

`-l` & Zeigt die Prozess-ID des Vaterprozess und den nice-Wert

Auf den System V-Maschinen verwendet man meist `-elf`. Die Bedeutung dieser Parameter ist in der Tabelle aufgeführt.

[Beliebte `ps`-Optionen (System V)][C]L Option & Anzeige

`-e` & Zeigt alle auf dem System laufenden Prozesse

`-l` & Zeigt Größe, Status und Priorität jedes Prozesses

`-f` & Zeigt User-ID, PID, PPID und Startzeit

Daneben gibt es eine große Menge anderer Optionen, die diverse Informationen zu den Prozessen liefern. Die Manpage von `ps` liefert eine vollständige Liste für Ihr System.

Die Titel der Prozessliste geben Auskunft über die angezeigten Informationen.

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	448	208	?	S	09:37	0:04	init [5]
root	2	0.0	0.0	0	0	?	SW	09:37	0:00	[keventd]
root	3	0.0	0.0	0	0	?	SW	09:37	0:00	[kapm-idled]
root	4	0.0	0.0	0	0	?	SWN	09:37	0:00	[ksoftirqd_CPU0]
root	5	0.0	0.0	0	0	?	SW	09:37	0:00	[kswapd]
root	6	0.0	0.0	0	0	?	SW	09:37	0:00	[bdf flush]
root	7	0.0	0.0	0	0	?	SW	09:37	0:00	[kupdated]
root	8	0.0	0.0	0	0	?	SW<	09:37	0:00	[mdrecoveryd]
root	11	0.0	0.0	0	0	?	SW	09:37	0:00	[scsi_ah_0]
root	13	0.0	0.0	0	0	?	SW	09:37	0:00	[khudb]
root	390	0.0	0.1	1396	628	?	S	09:39	0:00	/sbin/syslogd
root	393	0.0	0.3	1772	992	?	S	09:39	0:00	/sbin/klogd -c 1
root	399	0.0	0.2	2308	828	?	S	09:39	0:00	/usr/sbin/sshd
bin	452	0.0	0.1	1340	428	?	S	09:39	0:00	/sbin/portmap
arnold	1249	0.0	19.5	108924	62496	?	R	10:01	0:15	/opt/office52/soffice

arnold	2353	0.0	0.5	2840	1620	pts/4	S	17:11	0:00	/bin/bash
arnold	2416	0.0	0.5	2692	1716	pts/4	R	17:32	0:00	ps aux
arnold	2417	0.0	0.2	1976	824	pts/4	R	17:32	0:00	less

Dieser Ausschnitt der Prozessliste zeigt in der ersten Zeile den Prozess `init` mit der PID 1. Die erste Zeile zeigt in den Überschriften, was die einzelnen Spalten anzeigen. Diese Liste ist durch den Befehl `ps aux` entstanden. Je nach System und Parametern sind die Spalten anders. In Tabelle werden die wichtigsten Spalten beschrieben, die man durch `ps` anzeigen lassen kann. Welche Informationen durch welche Flags angezeigt werden, entnimmt man am besten der Manpage von `ps` auf dem jeweiligen System.

[Titelkürzel einer Prozessliste]L|L Kürzel & Beschreibung
 USER & Benutzer, der den Prozess gestartet hat
 PID & Prozess-ID, wird als Argument für `kill` verwendet
 PPID & Die Prozess-ID des Elternprozesses
 PGID & Prozessgruppen-ID
 SID & Session-ID
 PRI & Priorität des Prozesses. Je niedriger, desto mehr Rechenzeit
 NI & Nice-Wert oder SY für Systemprozesse
 %CPU & Anteil an der CPU-Auslastung
 %MEM & Anteil an der Speicherauslastung
 VSZ & Virtuelle Prozessgröße
 RSS & Größe des residenten Speichers
 TTY & Das Kontrollterminal des Prozesses
 STAT & Status des Prozesses, siehe unten
 SIZE oder SZ & Größe des Prozesses
 START oder STIME & Startzeitpunkt
 TIME & Die bisher verbrauchte CPU-Zeit
 WCHAN & Kernelfunktion, auf die der Prozess wartet
 COMMAND & Das Kommando, mit dem der Prozess gestartet wurde

Der Prozessstatus STAT kann verschiedene Zeichen haben. R bedeutet runnable. Der Prozess ist also gerade aktiv. Im Beispiel oben sind dies beispielsweise der `ps` mit der PID 2416, der die Anzeige erzeugt hat. Das immer wieder auftauchende S zeigt an, dass der Prozess schläft, also auf ein Ereignis wartet, das ihn weckt.

[Prozessstatus]C|L Status & Bedeutung
 R & Ausführbar
 S & Schlafend
 T & Gestoppt
 D & Auf der Festplatte wartend
 W & Prozess ist ausgelagert

Der Befehl `ps` kann von jedem Anwender aufgerufen werden, nicht nur von `root`. Dementsprechend ist es ein massives Sicherheitsloch, wenn ein Programm ein Passwort als Parameter im Klartext entgegennimmt.

Prozess-Hitparade: `top`

Während ps eine Momentaufnahme der Prozesse zeigt, stellt das Programm top die Prozesse in der Reihenfolge ihres CPU-Zeitverbrauchs dar. Wenn man also den Verdacht hat, dass ein Prozess überdurchschnittlich viel CPU-Zeit verbraucht, ist top ein ideales Werkzeug. Es erstellt eine Hitparade der Prozesse. Diejenigen Prozesse, die am meisten CPU-Zeit verbrauchen, stehen in der ersten Zeile. Hier folgt ein Beispiel für die Prozesstabelle, wie sie top anzeigt und regelmäßig aktualisiert.

```
11:32am up 3:04, 1 user, load average: 0.11, 0.16, 0.07
95 processes: 93 sleeping, 2 running, 0 zombie, 0 stopped
CPU states: 1.7% user, 2.3% system, 0.0% nice, 95.8% idle
Mem: 320136K av, 189100K used, 131036K free, 0K shrd, 5756K buff
Swap: 128512K av, 0K used, 128512K free 118592K cached
```

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	%CPU	%MEM	TIME	COMMAND
822	root	16	0	41876	8640	1720	S	1.9	2.6	0:39	x
1174	arnold	13	0	9180	9176	8256	R	1.1	2.8	0:01	kdeinit
1646	arnold	18	0	1004	1004	776	R	0.7	0.3	0:00	top
1	root	9	0	208	208	176	S	0.0	0.0	0:04	init
2	root	9	0	0	0	0	SW	0.0	0.0	0:00	keventd
3	root	9	0	0	0	0	SW	0.0	0.0	0:00	kapm-idled
4	root	19	19	0	0	0	SWN	0.0	0.0	0:00	ksoftirqd_CPU0
5	root	9	0	0	0	0	SW	0.0	0.0	0:00	kswapd
6	root	9	0	0	0	0	SW	0.0	0.0	0:00	bdfush
7	root	9	0	0	0	0	SW	0.0	0.0	0:00	kupdated
8	root	-1	-20	0	0	0	SW<	0.0	0.0	0:00	mdrecoveryd
11	root	9	0	0	0	0	SW	0.0	0.0	0:00	scsi_eh_0
13	root	9	0	0	0	0	SW	0.0	0.0	0:00	khudb
343	news	9	0	2772	2772	1284	S	0.0	0.8	0:00	innd
346	news	9	0	880	880	732	S	0.0	0.2	0:00	actived
367	news	9	0	704	704	596	S	0.0	0.2	0:00	overchan
370	news	9	0	2460	2460	1248	S	0.0	0.7	0:00	controlchan
373	root	9	0	0	0	0	SW	0.0	0.0	0:00	usb-storage-0

Mit h kann man sich die möglichen Kommandos anzeigen lassen. Aus top heraus kann man renice (siehe S. renice) durch die Taste r oder kill (siehe unten) über die Taste k aufrufen. Anschließend fragt das Programm, welcher Prozess gemeint war, und bittet um dessen PID. Mit der Taste q kann man das Programm wieder verlassen.

Das Programm top ist Bestandteil der Open Source-Systeme wie Linux oder FreeBSD und damit auch von MacOS X. Für die anderen Systeme kann es aber frei bezogen werden.

Quelle: <http://www.groupsys.com/top/index.html>

Unterabschnitte

- [SIGHUP](#)
 - [Priorität ändern statt töten](#)
 - [Kurzfristiges Parken](#)
 - [Terminieren](#)
-

Nicht immer mit Tötungsabsicht: kill

Das Kommando `kill` sendet Signale an einen Prozess. Wie der Name schon sagt, führt dies meist zum Dahinscheiden des Prozesses. Prozesse können aber viele Signale abfangen und verarbeiten. Dadurch ist eine Steuerung von außen möglich, was gerade für Hintergrundprozesse von Bedeutung ist. Auf diese Weise ist ein Booten zum erneuten Einlesen der Konfiguration unter UNIX nicht erforderlich. Damit ein Programm auf Signale reagiert, muss eine explizite Behandlung der Signale durch das Programm erfolgen (siehe S. `prgsignal`). Die Signale sind durchnummeriert, wenn auch nicht auf allen Systemen unbedingt gleich, und sie haben Namen. Beides kann als Parameter für `kill`, mit einem Minuszeichen gekennzeichnet, zur Spezifikation verwendet werden. Tabelle zeigt die meist verwendeten Signale und deren Nummer, sofern sie auf allen Systemen eindeutig ist.

[Signale]L|C|L Signal & Nr. & Bedeutung

SIGHUP & 1 & Sitzungsende. Signal an Dämonen, ihre Parameter-Datei neu lesen

SIGINT & 2 & Interrupt-Taste (Delete-Taste oder `ctrl-C`)

SIGKILL & 9 & Sofortiger Abschuss, kann vom Programm nicht abgefangen werden

SIGTERM & 15 & Aufforderung an den Prozess, sich regulär zu beenden.

SIGSTOP & & Hält den Prozess an, ohne ihn zu beenden.

SIGCONT & & Führt einen gestoppten Prozess fort

SIGHUP

Ursprünglich wurde das Signal SIGHUP versandt, wenn das Terminal ausgeschaltet wurde, auf dem eine Anwendung gestartet worden war. Auch in den Hintergrund gestellte Prozesse empfangen das Signal, wenn der Anwender, der sie gestartet hat, sich abmeldet. Will man dieses Verhalten unterbinden, muss dem Kommando der Befehl `nohup` vorangestellt werden. Das Signal kann zwar vom Programm abgefangen werden. Programme, die das Signal nicht explizit bearbeiten, werden beendet.

Das Signal SIGHUP hat seine wichtigste Anwendung darin, dass die meisten Dämonen auf ein `kill -1` ihre Konfiguration wieder neu lesen und sich entsprechend neu initialisieren. Aber nicht alle Hintergrundprozesse reagieren auf `kill -1`. So stirbt beispielsweise der `lpd`, wenn man ihn in der Hoffnung, er würde dann die `/etc/printcap` lesen, mit `kill -1` anschießt. In diesem Fall ist das aber auch kein Drama, da man nach Änderungen der `printcap` tatsächlich den `lpd` beenden und neu starten kann.

Priorität ändern statt töten

Der Fall, dass ein Administrator den Befehl `kill` tatsächlich zum Terminieren des Prozesses benutzt, ist tatsächlich eher selten. Ein Prozess, der von einem Anwender gestartet wurde, kann vom

Administrator nicht einfach beendet werden. Immerhin kann es sein, dass durch den Abschuss wichtige Daten verloren gehen. Erst wenn der Prozess durch extensiven CPU-Zeitverbrauch auffällt, wird man versuchen, etwas zu tun.

Der erste Versuch, einen solchen Amok laufenden Prozess zu besänftigen, wird darin bestehen, dessen Priorität mit `renice` herabzusetzen. Dadurch stört er immerhin nicht mehr andere Benutzer. Um den Prozess mit der PID 987 um 5 Level freundlicher zu gestalten, gibt man das Kommando:

```
renice +5 987
```

Kurzfristiges Parken

Hat man den Verdacht, dass dieser Prozess Schaden anrichtet oder ist `renice` nicht erfolgreich genug, kann man den Prozess mit dem Signal `SIGSTOP` anhalten und hat später die Möglichkeit, ihn mit `SIGCONT` wieder fortzusetzen. Gleichzeitig sollte man den Anwender verständigen, dass sein Prozess angehalten wurde, aber wieder aktivierbar ist.

```
kill -SIGSTOP 987
...
kill -SIGCONT 987
```

Terminieren

Erst wenn diese Möglichkeiten nicht mehr bestehen oder wenn Sicherheit darüber besteht, was der Prozess tut, oder falls Gefahr in Verzug ist, wird man den Prozess beenden. Um dies zu tun, verwendet man zunächst `SIGTERM`. Dies ist auch die Grundeinstellung von `kill`, wenn man keinen Parameter angibt. Das Signal `SIGTERM` wird von den meisten UNIX-Programmen abgefangen und führt dazu, dass der Prozess seine Daten sichert und geregelt endet. Beim Herunterfahren der Maschine bekommt jedes noch laufende Programm ein `SIGTERM` zugesandt. Dann hat das Programm fünf Sekunden Zeit, die Daten zu sichern. Anschließend kommt das `SIGKILL`, das auch dem störrigsten Prozess die Lampe ausbläst.

« [Prozessbeobachter](#) | [Informationen sammeln](#) | [Offene Dateien](#) »

Offene Dateien

Ob ein Prozess eine Datei geöffnet hat, ist eine wichtige Information. Eine geöffnete Datei kann nicht einfach gelöscht werden, und ein schreibender Prozess sollte nicht einfach abgebrochen werden. Es gibt also einen Bedarf an Information, welche Datei von welchen Prozessen bearbeitet wird und welcher Prozess welche Dateien geöffnet hat.

Um festzustellen, welche Prozesse auf eine Datei zugreifen, gibt es den Befehl `fuser`. Er zeigt die PID der Prozesse, die die als Parameter angegebene Datei geöffnet haben:

```
gaston> fuser unix.ps
unix.ps:      1176  1190
```

Wendet man `fuser` auf ein Verzeichnis an, kann man sehen, welche Prozesse unterhalb dieses Verzeichnisses gestartet worden sind. Das ist wichtig, wenn man beispielsweise ein Dateisystem per `umount` ausklinken möchte und die Meldung erscheint, dass das Gerät noch benutzt wird.

Die andere Frage, nämlich welche Dateien ein bestimmter Prozess geöffnet hat, kann man mit Hilfe des Programmes `lsof` (list open files) beantwortet werden. Es liefert alle offenen Dateien des Systems. Darin sind auch die Netzwerkverbindungen enthalten. Hier sehen Sie eine Beispielausgabe:

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE	NODE	NAME
init	1	root	mem	REG	3,5	333780	178159	/sbin/init
portmap	386	root	mem	REG	3,5	28184	178286	/sbin/portmap
portmap	386	root	mem	REG	3,5	342535	275298	/lib/ld-2.1.3.so
portmap	386	root	mem	REG	3,5	44729	275322	/lib/libutil.so.1
portmap	386	root	mem	REG	3,5	4070406	275303	/lib/libc.so.6
syslogd	404	root	mem	REG	3,5	29252	485827	/usr/sbin/syslogd
syslogd	404	root	mem	REG	3,5	342535	275298	/lib/ld-2.1.3.so
syslogd	404	root	mem	REG	3,5	4070406	275303	/lib/libc.so.6

In der zweiten Spalte steht die PID, die man leicht per `grep` oder besser per `awk` herausfiltern kann. Der folgende Aufruf liefert alle Dateien, die der Prozess mit der PID 719 geöffnet hat.

```
lsof | awk '{if ( $2==719 ) print $9 }'
```

Da nicht jeder Administrator die Syntax von `awk` auswendig kennt, funktioniert natürlich auch die einfachere Variante mit `grep`:

```
lsof | grep 719
```

Der Nachteil dieser unsportlichen Lösung ist natürlich, dass nun auch Zeilen herausgegriffen werden, die an anderer Stelle als in Spalte 2 die Zahl 719 haben.

Falls es einmal notwendig wird, Prozesse zu terminieren, die noch offene Dateien halten, kann man versuchen, den Schaden zu begrenzen, indem man das System auffordert, die Daten in den Puffern

auf die Platte zu schreiben. Der Aufruf `sync` erreicht dies. Er braucht keine weiteren Parameter.

« [Nicht immer mit Tötungsabsicht: Informationen sammeln](#) | [Programmzusammenbrüche \(Core Dump\)](#) »

Programmzusammenbrüche (Core Dump)

Wenn ein Programm unter UNIX Dinge tut, die es nicht darf, dann wird ihm vom System einfach die Betriebserlaubnis entzogen. Zu diesen unerlaubten Beschäftigungen gehört beispielsweise der Griff in den Speicher anderer Prozesse. UNIX teilt den Prozessen klar abgegrenzte Speicherbereiche zu und merkt, wenn ein Prozess in die Systembereiche eindringen möchte. In einem solchen Fall sendet UNIX dem Prozess ein Signal, im letzten Fall SIGSEGV, eine segment violation. Man könnte das frei mit »Grenzverletzung« übersetzen.

Wenn UNIX ein Programm auf diese Weise beendet, schreibt es den Speicherbereich des Prozesses in eine Datei namens core, die im aktuellen Arbeitsverzeichnis angelegt wird. Da zum Speicher auch der Stack gehört, kann ein Debugger (siehe S. debug) feststellen, in welcher Funktion der Zusammenbruch erfolgte. Diese Information kann für den Programmierer eine große Hilfe sein.

Wenn man als Administrator eine Datei namens core findet, ist zunächst deren Alter interessant. Das Alter gibt Auskunft über den Zeitpunkt des Zusammenbruchs. Ist der verantwortliche Programmierer greifbar, wird er sich für diese Datei interessieren. Ansonsten gehören core-Dateien eher zum lästigen Abfall, der bei der Entwicklung von Software nun einmal entsteht. Darum steht bei einigen Maschinen eine Anweisung in der crontab, alle älteren Dateien namens core zu suchen und zu löschen. Auf einer Produktionsmaschine sollten Core Dumps nicht entstehen. Insofern wäre eine Beseitigung solcher Spuren eines Zusammenbruchs nicht sehr sinnvoll.

Systemabsturz (Kernel-Panic)

Eine Kernel-Panic ist ein Fehler, der so heftig ist, dass das Betriebssystem alle Tätigkeiten einstellt. Die letzte Aktion besteht darin, einen Speicherabzug in die Swap-Partition zu schreiben. Als Ursache einer Kernel-Panic kommen im Normalfall eigentlich nur Probleme mit der Hardware oder mit deren Treibern in Frage. Anwendungsprogramme haben keinen Zugriff auf Bereiche, die das System derart außer Tritt bringen können. Selbst Dämonen und Systemprozesse würden im Falle eines Amoklaufs auf die Ausnahmebehandlung des Betriebssystems stoßen und einen schnellen Tod finden.

Der Kernel

Der Kernel von UNIX könnte als das Hauptprogramm des Betriebssystems bezeichnet werden. In der klassischen Variante enthält er alles, was zum Betriebssystem gehört. Vom Prozessmanager (dem Scheduler), über die Speicherverwaltung, das Dateisystem bis hin zu den Gerätetreibern war alles ein einziges Programm. In den Anfangstagen von UNIX lag das Betriebssystem als Source vor, und es wurde vom Administrator an die Bedürfnisse seiner Umgebung angepasst. Anschließend wurde es mit dem mitgelieferten C-Compiler übersetzt. Die Konstanten, die das Laufzeitverhalten prägten, wurden an die Einsatzumgebung der Maschine angepasst, die benötigten Treiber wurden ausgewählt, und danach wurde ein Kernel gebildet. Eigentlich müsste es korrekt »Kernel bauen« heißen, da es sich vom englischen Wort build herleitet. Inzwischen ist der Begriff »bilden« bereits so verbreitet, dass er nur noch Administratoren mit abgebrochenen Germanistikstudium stört. Dieser Kernel wurde dann in das Rootverzeichnis gelegt.

Fast alle großen Hersteller von UNIX liefern heute weder die Quelltexte aus, noch ist ein Compiler im Paket enthalten. Das hat mehrere Gründe. Zum einen ist die Art der Anpassung nicht gerade besonders komfortabel. Das Übersetzen ist eine langwierige Geschichte. Das Ausliefern des Quelltextes möchte man vermeiden, um der Konkurrenz keinen Einblick in die eigenen Fortschritte zu geben, und schließlich erhofft man sich durch den Verkauf des Compilers als separates Paket ein Zusatzgeschäft.

Die meisten Parameter des Betriebssystems können heute dem Betriebssystem mitgeteilt werden, ohne eine Neukompilierung durchzuführen. Einige Größen, wie beispielsweise die Anzahl der Plattenpuffer, werden nicht mehr festgelegt. Stattdessen stellt das Betriebssystem Puffer dynamisch zur Verfügung, wenn sie erforderlich sind. Treiber werden nicht mehr in den Kernel integriert, sondern als Module während des Laufs hinzugezogen. Das Bilden des Kernels wird darum auf den meisten UNIX-Plattformen durch ein menügesteuertes Systemtool erreicht und hat mit dem Kompilieren früherer Zeiten nicht mehr viel zu tun.

Linux wird, allerdings aus anderen Gründen, heute noch mit dem kompletten Quellcode ausgeliefert. Auch ein Compiler wird mitgeliefert, sodass das Bilden des Kernels durch mehrere Aufrufe von make (siehe S. make) abläuft. Nur wenige Parameter werden allerdings noch durch das Ändern von Konstanten im Quellcode angepasst. Normalerweise sind solche Parameter durch Umgebungsvariablen einstellbar. Auch die meisten Treiber werden als Module beim Start geladen. Nur bei einigen exotischen Geräten muss man noch selbst Treiber aktivieren oder deaktivieren. Durch die Möglichkeit, Fähigkeiten des Kernels wegzulassen, kann man allerdings sehr spezialisierte, kleine Kernel bauen, die auch auf leistungsschwacher Hardware noch laufen.

Generieren eines Linux-Kernels

Das Generieren des Kernels erfolgt durch Aufrufe des Programmes make mit unterschiedlichen Zielen.

Mit `make config` wird ein Dialog gestartet, in dem man wie in einem Multiple-Choice-Test beantwortet, welche Bestandteile der neue Kernel haben soll. Nacheinander werden die Bestandteile angeboten. Der Administrator muss entscheiden, ob er die jeweilige Fähigkeit einbinden will oder nicht, oder ob sie als Modul zur Verfügung stehen soll. Es werden schlüssige Voreinstellungen

vorgeschlagen, und zur Unterstützung gibt es kleine Hilfetexte zu jedem Thema.

Der Aufruf von `make dep` ermittelt die Abhängigkeiten anhand des Konfigurationslaufs und stellt zusammen, welche Dateien für den Kernel gebraucht werden.

Ein `make bzImage` erzeugt schließlich den Kernel und generiert auch gleich den passenden Booteintrag. Wenn man sich noch nicht so ganz sicher ist, kann man auch erst einmal eine Diskettenversion erzeugen. Der Befehl lautet dann `make bzdisk`. Vorher sollte eine formatierte Diskette ins Laufwerk gesteckt werden.

Basteleien am Kernel sind durchaus riskant. Die Mindestsicherheitsvorkehrungen sind eine Sicherungskopie des bisherigen, funktionierenden Kernels und ein Medium (CD oder Diskette), mit dem man das System so weit booten kann, um den gesicherten Kernel wieder an seine ursprüngliche Position zu bringen.

Unterabschnitte

- [Generieren eines Linux-Kernels](#)
- [Dynamische Bibliotheken](#)
- [Module](#)

« [Systemabsturz \(Kernel-Panic\)](#) | **[Administration](#)** | [Dynamische Bibliotheken](#) »

Dynamische Bibliotheken

Bestimmte immer wieder benutzte Bibliotheken wurden früher statisch zu jedem UNIX-Programm hinzugebunden. Da es Speicherplatzverschwendung ist, die gleiche Bibliothek in jedem Programm wieder und wieder auf der Platte und hinterher auch im Hauptspeicher zu haben, gibt es dynamische Bibliotheken. Besonders mit der Verwendung grafischer Oberflächen ist es fast unmöglich, alle Bibliotheken statisch zu jeder Anwendung hinzuzuladen.

Im Gegensatz zu anderen Systemen ist es nicht üblich, dass Anwenderprogramme neue Versionen der von ihnen benötigten dynamischen Bibliotheken heimlich installieren. Das Update der Bibliotheken obliegt dem Administrator, und das Programm kann nur seinen Kummer darüber zum Ausdruck bringen, dass eine bestimmte Mindestversion einer Bibliothek nicht vorliegt, und gegebenenfalls enden. Damit ist auf einfache Weise verhindert, dass Software nicht mehr läuft, weil ein anderes Programm installiert wurde.

Man erkennt dynamische Bibliotheken an der Namensendung .so. Sie befinden sich im Normalfall im Verzeichnis /lib oder /usr/lib.

Module

Ursprünglich wurden alle Treiber in den Kernel eingebunden. Wenn es notwendig war, eine andere Konfiguration zu verwenden, wurde halt ein neuer Kernel generiert. Solaris, Linux, FreeBSD und damit MacOS X besitzen ein Modulkonzept, um Treiber separat vom Kernel installieren zu können.

Unter Solaris erhält man mit dem Befehl `modinfo` eine Übersicht über die aktuell geladenen Module. Mit `add_drv` kann man Treiber hinzufügen und mit `rem_drv` wieder entfernen. Nach dem Hinzufügen wird durch den Aufruf von `drvconfig` eine Neukonfiguration des Verzeichnisses `/devices` durchgeführt. Für Module, die keinen Zugriff auf Gerätedateien brauchen, lauten die Befehle `modload` und `modunload`. vgl. Nemeth, Evi / Snyder, Garth / Seebass, Scott / Hein, Trent R.: UNIX Systemverwaltung. Markt+Technik - Prentice Hall, 2001, S. 334f.

FreeBSD verwendet die Kommandos `kldload` und `kldunload` zum Einbinden und Entfernen von Modulen. `kldstat` gibt eine Übersicht. MacOS X erbt diese Eigenschaften von FreeBSD und nennt die Kommandos `kmodload`, `kmodunload` und `kmodstat`.

Linux braucht zwingend in seinem Kernel den Treiber für die Hardware und für das Dateisystem, auf dem sich das Wurzelverzeichnis befindet, damit das Booten möglich ist. Alle anderen Bestandteile können auch als Module geladen werden. Die Module befinden sich im Verzeichnis `/lib/modules`. Darunter befindet sich ein Verzeichnis, das die Versionsnummer des Kernels trägt. Darunter liegt ein Verzeichnisbaum, in dem die Module themenspezifisch abgelegt sind.

Der Befehl `lsmod` zeigt die geladenen Module an. Er zeigt den Namen des Moduls, seine Größe und die Anzahl der Zugriffe auf das Modul. Nur wenn die Zugriffe 0 sind, kann ein Modul mit dem Befehl `rmmod` wieder entfernt werden. Mit dem Befehl `insmod` kann ein Modul geladen werden. Mit weiteren Optionen können dem Modul weitere Parameter übergeben werden, beispielsweise Interruptnummern oder I/O-Adressen. Eine Variante von `insmod` ist der Befehl `modprobe`. Er versucht, ein Modul zu installieren, und kann anhand der Datei `/etc/modules.conf` feststellen, wie ein Modul einzubinden ist.

In der Datei `/etc/modules.conf` stehen die Informationen zu den verschiedenen Modulen. Die Datei hat verschiedene Einträge. Zunächst kann sie die Einträge in `/dev` auf Namen mit `alias`-Anweisungen umsetzen. Beispielsweise:

```
# block dev aliases
alias block-major-1 rd
alias block-major-2 floppy
alias block-major-3 off
alias block-major-7 loop
...
alias char-major-6 lp
alias char-major-9 st
```

Mit der Anweisung `options` können einem Modul Parameter übergeben werden. Im folgenden Beispiel werden der Soundkarte `cs4232` eine IO-Adresse, ein Interrupt und DMA-Kanäle mitgegeben:

```
options cs4232 io=0x534 irq=5 dma=1 dma2=0 mpuio=0x330 mpuiirq=9
```

Mit den Anweisungen post-install und pre-install kann festgelegt werden, welche Treiber vorher oder nachher installiert oder deinstalliert werden müssen.

« [Dynamische Bibliotheken](#) | [Der Kernel](#) | [Netzwerk](#) »

Netzwerk

- Netzwerk
 - Client-Server-Architekturen
 - TCP/IP, der Standard
 - Routing: Verbindung mehrerer Netzwerke
 - Namensauflösung
 - Next Generation IPv6
 - Grundausrüstung Bordwerkzeug
 - TCP/IP-Dienste
 - Allgemeines zum Internetanschluss
 - Dynamische TCP/IP-Nummern (DHCP)
 - E-Mail
 - Newsgroups
 - Jeder Rechner ein eigener Webserver
 - Fremdgegangen - Andere Protokolle
 - Firewall und Masquerading
 - Proxy
-

Netzwerk

Die Zeit der zentralen Großrechner mit den passiven Terminals klingt aus. Gerade UNIX findet heute seinen Haupteinsatz als leistungsfähiges und zuverlässiges Serversystem in einer Netzwerkumgebung.

Ein Netzwerk ist zunächst nichts anderes als die Verbindung mehrerer Computer per Kabel, über das sie miteinander kommunizieren können. In Kombination mit geeigneter Software können Sie von Ihrem Arbeitsplatz aus Ressourcen anderer Computer nutzen. Am bekanntesten dürfte die Möglichkeit sein, auf die Platte oder den Drucker gemeinsam zugreifen zu können. Das Motiv ist nicht allein Sparsamkeit. Netzwerke ermöglichen das Teilen von Daten bei höchster Aktualität.

Auf den ersten Blick scheint es, als würde ein Netzwerk Objekte wie Drucker oder Platten zur Verfügung stellen. Bei näherer Betrachtung handelt es sich aber um Dienste (engl. service). So werden die Druckdaten nicht an den Netzwerkdrucker selbst gesendet, sondern an einen Prozess. Dieser veranlasst dann als Dienstleister (engl. server) den Druck. Da die Druckdaten nicht direkt an den Drucker gehen, kann der Prozess vor dem Druck die Berechtigungen zu prüfen. Der teure Farblaserdrucker der Werbeabteilung soll beispielsweise nicht allen Angestellten zur Verfügung stehen, um ihre Urlaubsfotos auszudrucken.

Das größte aller Netze ist das Internet. Hier gibt es abertausende von Dienst Anbietern. Neben dem World Wide Web bietet es eine schnelle, kostengünstige Kommunikation per E-Mail. Darum ist eine Firma daran interessiert, vielen Arbeitsplätzen einen Zugang zu dieser Informationsquelle zu ermöglichen. Auf der anderen Seite besteht auch das Risiko, dass das eigene Netz über das Internet ausgespäht oder angegriffen wird.

Jedes Netzwerk braucht ein Protokoll, in dem festgelegt wird, welcher Teil der Nachricht Adresse, Absender, Kontrollinformation bzw. Daten ist. Inzwischen ist TCP/IP (Transmission Control Protocol/ Internet Protocol) das unangefochten wichtigste Protokoll. TCP/IP ist für UNIX nicht nur das Zugangsprotokoll zum Internet, sondern auch die Basis für lokale Netzwerke. Dabei spielen die vom PC her bekannten Platten- und Druckserver keine so große Rolle wie das Verteilen von Anwendungen in Client-Server-Architekturen. In UNIX-Netzen geht es mehr um das Starten von Prozessen auf entfernten Maschinen oder um das Verteilen von Prozessen auf mehrere Maschinen.

In TCP/IP-Umgebungen finden Sie selten ausschließlich UNIX-Maschinen. So haben Sie immer wieder damit zu tun, auch MS Windows oder MacOS zur Zusammenarbeit zu bewegen. Sie werden oft als Front-End benutzt, während das Back-End unter UNIX läuft.

-
- [Client-Server-Architekturen](#)
 - [Ethernet als Verkabelungsbeispiel](#)
 - [Pseudoschnittstelle loopback](#)

- Pakete in Paketen
- TCP/IP, der Standard
 - Die TCP/IP-Nummer
 - Das Prüftool ping
- Routing: Verbindung mehrerer Netzwerke
 - Gateways
 - Statische Festlegung einer Route
 - Statisches Routing: Ein Beispiel
 - Subnetze
 - Dynamisches Routen
 - CIDR - Classless Inter-Domain Routing
- Namensauflösung
 - Der Host- und Domainname
 - Die Datei /etc/hosts
 - Die Datei /etc/services
 - Domain Name Service: DNS
 - Network Information Service: NIS
 - Netzgruppen: /etc/netgroup
- Next Generation IPv6
- Grundausrüstung Bordwerkzeug
 - ICMP und ping
 - Verbindung zwischen Prozessen: netstat
 - Anzeigen der Netzwerkadapter
 - Anzeigen der Routingtabelle
 - Routen verfolgen: traceroute
 - HP-UX: lanadmin
- TCP/IP-Dienste
 - inet-Dämon
 - File Transfer Protocol (FTP)
 - Anonymer FTP-Server
 - TFTP, schnell und vertrauensvoll
 - Terminaldienst (telnet)
 - r-Kommandos
 - Wenn Sicherheit vorgeht: die ssh und scp
 - NFS - Network File System
 - Automatisches Mounten
- Allgemeines zum Internetanschluss
- Dynamische TCP/IP-Nummern (DHCP)
- E-Mail
 - Format einer E-Mail
 - UNIX und Mail
 - SMTP (Simple Mail Transport Protocol)
 - Mailqueue
 - Verteilen der Post: sendmail -q

- Weiterleiten der Post: aliases und forward
- Lokale Mail lesen
- POP3
- IMAP
- Post sammeln: fetchmail
- Mailserver und Domain
- Erstes Beispiel: Interne Firmenmail
- Zweites Beispiel: Anbindung an das Internet
- Newsgroups
 - News lesen
 - Installation des Newsservers inn
 - Beispiel: Newsserver zur Projektverwaltung
 - Gruppen anlegen
 - Verbindung nach außen
 - Newsgroups saugen
 - NNTP Protokollbeschreibung
- Jeder Rechner ein eigener Webserver
 - Hypertext und HTML
 - Start des Servers
 - Die Konfigurationsdatei httpd.conf
 - Privatadministration per .htaccess
 - Kommunikation per HTTP
 - Virtuelles Hosting
 - CGI: der Server schlägt zurück
- Fremdgegangen - Andere Protokolle
 - Samba: UNIX im Windows-Netz
 - Novell-Zugriffe
 - Mac im Netz: netatalk
- Firewall und Masquerading
 - Funktionsweise einer Firewall
 - Masquerading
- Proxy

Client-Server-Architekturen

Der Zweck eines Netzes ist es, einem Prozess auf einem Rechner mit Informationen zu versorgen, die auf einem anderen Computer vorhanden sind. Der Auslöser ist also immer ein Prozess, der eine Information anfragt und auf die Antwort wartet. Einen Anfrager nennt man Client, einen Antworter nennt man Server. Ein Server ist also in erster Linie ein Prozess und kein Computer. Derselbe Computer kann durchaus gleichzeitig als Client und als Server auftreten, indem er bestimmte Anfragen beantwortet und auf der anderen Seite Anfragen stellt.

Eine Software, die nach dem Client-Server-Prinzip arbeitet, ist auf zwei Seiten aufgeteilt. Sie ist quasi irgendwo »durchgesägt«. Das Front-End läuft auf dem Arbeitsplatzrechner, und ein anderer Teil arbeitet auf einem anderen, typischerweise zentralen Rechner, der dann auch meist als Server bezeichnet wird, weil auf ihm die Serverprozesse laufen.

Unterabschnitte

- [Ethernet als Verkabelungsbeispiel](#)
- [Pseudoschnittstelle loopback](#)
- [Pakete in Paketen](#)

Ethernet als Verkabelungsbeispiel

Bei der Verkabelung wird heutzutage meistens Ethernet eingesetzt. Das ist für lokale Netzwerke inzwischen nahezu konkurrenzlos. In seiner einfachsten Form besteht ein Ethernet aus einem Koaxialkabel. Ein Koaxialkabel besteht aus einem Draht, der von einer Abschirmung umgeben ist. Antennenkabel sind typischerweise auch Koaxialkabel, und je einem Widerstand an jedem Ende. Ein Computer wird mit dem Kabel durch einen Abgriff verbunden. Beim dem in kleinen Netzwerken teilweise noch verwendeten Koaxialkabel RG58 ist das ein auf dem BNC-Stecker basierendes T-Stück. Dieses T-Stück steckt direkt auf einem Transceiver am Ethernetcontroller.

Inzwischen verwendet man auch in kleinen Netzen mehr und mehr eine Twisted-Pair-Verkabelung. Das Kabel besteht, wie der Name schon sagt, aus verdrehten Drähten. Genauer gesagt führen in einem Twisted Pair Kabel zwei Drähte zum Computer und zwei zurück zum Hub. Ein Hub ist der Rückgrat des Netzes. Denn obwohl das Ganze so aussieht, als wäre die Verkabelung sternförmig angeordnet, wird vom Hub jeder Abgang schleifenartig zu einem langen Draht verkoppelt, sobald das Kabel auf der anderen Seite durch den Computer verbunden wird. Der Vorteil von Twisted Pair liegt vor allem in der Robustheit. Das Koaxialkabel RG58 neigt an den Anschlüssen zu Wackelkontakten, insbesondere bei den Abschlusswiderständen an den Kabelenden.

Jeder Ethernetcontroller hat seine eigene, weltweit eindeutige Nummer, die 48 Bit groß ist. Diese Nummer ist meist in das ROM des Controllers gebrannt. Bei Sun Maschinen befinden sie sich in einem batteriegepufferten RAM. Darum ist es eine gute Idee, sich diese Nummer zu notieren. Die Batterie könnte ja mal schlappmachen. Die Nummer wird beim Booten angezeigt. Der Ethernetcontroller lauscht die ganze Zeit am Kabel und sobald ein Paket kommt, das die Nummer des Controllers als Adresse hat, holt er es in seinen Speicher und gibt es an das Betriebssystem weiter.

Will der Controller selbst Daten senden, packt er sie in Pakete zu je maximal 1500 Byte und setzt die Ethernetadresse des Zielcomputers, gefolgt von der eigenen Adresse als Absender hinein. Zu guter Letzt enthält jedes Paket eine Prüfsumme, die CRC (Cyclic Redundancy Check) und schon ist das Paket sendebereit.

Zum eigentlichen Senden wartet der Controller ab, bis sich auf dem Netzwerk eine Sendepause ergibt. Nun beginnt er damit, ein paar alternierenden Folgen von Einsen und Nullen dem Paket voranzusenden, und lauscht, ob diese von einem anderen Controller gestört werden, der vielleicht im gleichen Moment versucht zu senden. Ist das der Fall, versuchen es beide Controller später noch einmal. Geht alles glatt, setzt er Paket um Paket auf diese Weise hinterher, bis seine Daten übermittelt sind.

Dieses Verfahren, mit dem Kabel umzugehen nennt sich Carrier Sense Multiple Access Bus with Collision Detect mit dem Kürzel CSMA/CD. Es bedeutet soviel wie »Abtasten des Mediums mit mehrfachem Zugriff bei Kollisionserkennung«. Die Kollision von Paketen gehört also zum Protokoll und ist kein Drama. Allerdings häufen sich die Kollisionen bei höherer Netzlast und können dann zu Problemen mit dem Durchsatz führen.

Wer sich für Details interessiert, was wirklich auf Controller-Ebene passiert, wie die Pakete genau aussehen, wie die Prüfsummen arbeiten, der sei hier auf die Werke von Tanenbaum Andrew S.

Tanenbaum: Computer Networks. Prentice Hall, Englewood Cliffs, 1987. und Comer Douglas E. Comer: Internetworking with TCP/IP. Prentice Hall, 2nd ed., 1991 verwiesen.

« [Client-Server-Architekturen](#) | **Client-Server-Architekturen** | [Pseudoschnittstelle loopback](#) »

Pseudoschnittstelle loopback

TCP/IP wird nicht nur verwendet, um mit anderen Rechnern Kontakt aufzunehmen. Manchmal führt der Rechner auch Selbstgespräche. So arbeitet die grafische Oberfläche von UNIX über TCP/IP. Dabei geht die Kommunikation auf modernen Workstations vom eigenen Rechner zum eigenen Display. Bei Maschinen, die überhaupt keine Netzwerkanschlüsse haben, wird eine Schnittstelle gebraucht, über die der Rechner mit sich selbst Kontakt aufnehmen kann. Diese Schnittstelle nennt sich loopback, weil sie eine Schleife auf den Rechner selbst zurückführt.

Pakete in Paketen

Auf der Basis von Ethernetpaketen werden TCP/IP-Pakete versandt. Als Adresse werden IP-Nummern und der Port verwendet (siehe unten). Dabei kennt ein Ethernetcontroller nur Ethernetnummern. Die IP-Nummern müssen also auf die Ethernetnummern abgebildet werden. Dieses Protokoll nennt sich ARP (Address Resolution Protocol) und ist eigentlich unterhalb des TCP/IP anzusiedeln.

Es kann ein Problem auftreten, wenn eine Netzwerkkarte zwischen Rechnern getauscht wurde. Da die IP-Nummer nun unter einer anderen Ethernetnummer zu finden ist, ist die ARP-Tabelle nicht mehr gültig. Nach dem Tausch läuft diese IP-Nummer bei anderen Rechnern im Netz noch unter einer anderen Ethernetnummer oder umgekehrt. Die ARP-Tabelle kann unter UNIX mit dem Befehl `arp` bearbeitet werden. Mit der Option `-a` wird die Tabelle angezeigt, mit `-d` wird gelöscht und mit `-s` kann ein Eintrag gesetzt werden. Näheres finden Sie in der Manpage von `arp`.

« [Pseudoschnittstelle loopback](#) | [Client-Server-Architekturen](#) | [TCP/IP, der Standard](#) »

TCP/IP, der Standard

Unterabschnitte

- [Die TCP/IP-Nummer](#)
 - [Netzwerkklasse und Netzmaske](#)
 - [Freie IP-Nummern](#)
 - [Grundeinstellungen des Netzadapters: ifconfig](#)
 - [Dauerhaftes Einstellen der IP-Nummer](#)
 - [Das Prüftool ping](#)
-

Unterabschnitte

- [Netzwerkklasse und Netzmaske](#)
 - [Freie IP-Nummern](#)
 - [Grundeinstellungen des Netzadapters: ifconfig](#)
 - [Dauerhaftes Einstellen der IP-Nummer](#)
-

Die TCP/IP-Nummer

Die TCP/IP- oder Internetnummer ist eine 32-Bit Zahl, die die Netzwerkschnittstelle eines Computers im Netz eindeutig bestimmt. Der erste Teil der Nummer bezeichnet das Netz, in dem der Computer sich befindet, der zweite Teil den Computer selbst. Alle Computer, die direkt miteinander verbunden sind, gehören zum gleichen Netz und haben die gleiche Netzkennung in ihrer Internetnummer. Wie Sie mit einem Computer kommunizieren können, der eine fremde Netzkennung hat, wird beim Thema Routing (siehe ab S. routing) behandelt. Ohne besondere Maßnahmen reagiert der Computer auf den Versuch, auf eine fremde Netzadresse zuzugreifen mit der Fehlermeldung »no route to host«.

Der hintere Teil der Internetnummer ist die Adresse des Computers im Netz. Um genau zu sein, ist es die Kennung des Netzwerkadapters des Computers. Da aber die meisten Computer nur einen Adapter haben, soll der einfacheren Lesbarkeit halber auch weiter von Computern die Rede sein. Jeder Adapter muss eine eigene Nummer haben. Mehrere Rechner mit gleicher Hostkennung führen in größeren Netzen zu schwer auffindbaren Fehlern.

Die Grenze zwischen Netzkennung und Hostkennung liegt normalerweise auf einer Bytegrenze. Darum wird für IP-Nummern eine byteweise Darstellung gewählt. Da Dezimalzahlen am einfachsten zu lesen sind, schreibt man jedes der vier Byte einer IP-Adresse dezimal auf und trennt sie durch einen Punkt. Beispielsweise hat mein Arbeitsplatzrechner in meinem heimatlichen Netzwerk die TCP/IP-Nummer 192.168.109.144. Hexadezimal ergibt dies C0 A8 6D 90. Wieder als ganze Zahl geschrieben lautet die Nummer 3232263568. Mein Macintosh hat die Adresse 192.168.109.25 oder hexadezimal C0 A8 6D 19 und damit 3232263449.

[Verschiedene Darstellungen von IP-Nummern]
L|L|L Darstellung & Arbeitsplatz & Macintosh
dezimal & 3232263568 & 3232263449
hexadezimal & C0 A8 6D 90 & C0 A8 6D 19
dotted decimal & 192.168.109.144 & 192.168.109.25

Netzwerkklasse und Netzmaske

Wie schon ausgeführt, kennzeichnet die IP-Nummer den Rechner bzw. dessen Netzadapter und das Netzwerk, in dem sich der Rechner befindet. Im Beispiel meines Netzes ist offensichtlich 192.168.109 bei beiden Rechnern gleich. Und tatsächlich benennen die ersten drei Byte in diesem Fall das Netz und das letzte Byte den Rechner. Man sollte also denken, ich könnte in mein Hausnetz maximal 256 Rechner einbinden, da ein Byte die Werte von 0 bis 255 annehmen kann. Allerdings sind die 0 und die

255 für andere Zwecke reserviert, so dass ich mit 254 Rechnern in meinem Arbeitszimmernetz auskommen muss.

Welcher Teil der IP-Nummer zum Host und welche zum Netzwerk gehört, wird durch die Netzmaske bestimmt. Die Maske ergibt sich zunächst aus der Netzklasse, zu der die IP-Nummer gehört. Die Netzklasse wiederum wird durch die ersten Bits des ersten Bytes der IP-Nummer bestimmt.

Ist das erste Bit 0, gehört die Adresse zu einem Class A Netz. Betrachtet man das erste Byte in binärer Darstellung, schaut sich also die einzelnen Bits an, so entscheidet das erste Bit darüber, ob die Zahl größer oder kleiner als 128 ist. Ist das erste Byte einer Netzwerkadresse kleiner oder gleich 127, gehört sie als zu einem Class A Netz. Class A Netze haben eine Netzmaske von 255.0.0.0.

Warum heißt nun diese Zahl Netzmaske und warum ist sie im ersten Byte 255? Mit der Netzmaske und der UND-Verknüpfung kann man den Netzanteil einer IP-Adresse herausfiltern. UND ist ein logischer Ausdruck auf Binärebene, die genau dann 1 ergibt, wenn beide Operanden 1 sind. Um das an einem Beispiel zu demonstrieren, wird die Class A Netzadresse 10.3.4.7 mit der Netzmaske 255.0.0.0 gefiltert.

Adresse:				
dezimal	10	3	4	7
binär	00001010	00000011	00000100	00000111

Netzmaske:				
dezimal	255	0	0	0
binär	11111111	00000000	00000000	00000000

Adresse	00001010	00000011	00000100	00000111
Netzmaske	11111111	00000000	00000000	00000000
UND	-----			
Ergebnis	00001010	00000000	00000000	00000000
dezimal	10	0	0	0

Durch die Verknüpfung UND von Adresse und Netzmaske wird der Netzanteil der Adresse »herausmaskiert«. Der Netzanteil heißt hier 10.0.0.0. Das erste Byte ist die Netzadresse und der Rest die Adresse des Netzadapters.

Die 192 ist binär 11000000. Alle Werte, die kleiner als 192 sind, beginnen mit der Bitkombination 10 und gehören zu einem Class B Netz. Die Netzmaske im Class B Netz ist 255.255.0.0, also sind die ersten zwei Byte der Anteil der Netzadresse und der Rest ist die Rechneradresse.

Die Nummern 224-255 im ersten Byte sind reserviert und dürfen nicht als Netzwerknummern verwendet werden. Dies sind die IP-Nummern, die mit drei Einsen beginnen. Das erste Byte in einem Class C Netz beginnt mit der Bitkombination 110 und liegt zwischen 192 und 223 (also auch mein Hausnetz). Es hat die Netzmaske 255.255.255.0 und hat damit 3 Byte für die Netzkennung und ein Byte für die verschiedenen Computer.

Bei einem größeren Unternehmen könnte ein Class B Netz mit den Nummern 128-191 schon sinnvoller sein, da hier ca. 16000 Computer im Netz sein können. Richtig viele Computer passen in ein Class A Netz. Allerdings gibt es von diesen nur 128.

[Zahlen in den Klassen]L|R|R|R|R|L & Anzahl Bits & Anzahl & Anzahl Bits & Anzahl der & Kennungen
& für Netze & der Netze & für Hosts & Hosts & des ersten Byte
Class A & 7 & 128 & 24 & 16777216 & 1-127 (0xxxxxx)
Class B & 14 & 16384 & 16 & 65536 & 128-191 (10xxxxxx)
Class C & 21 & 2097152 & 8 & 256 & 192-223 (110xxxxx)

Die Nummer einer Station darf weder alle Bits der Host-ID 1 oder 0 haben. Alle Bits 1 ist die Broadcast-Adresse des Netzes. Ein Paket an diese Adresse wird von allen Rechnern des Netzes gelesen. Sind alle Bits 0, bezeichnet diese Adresse das Teilnetz. Dies wird beispielsweise beim Routing so verwendet.

Wenn also zwei oder mehrere Rechner in einem lokalen Netz direkt miteinander verbunden werden sollen, müssen folgende Dinge gewährleistet sein:

1. Der Netzwerkteil der IP-Nummern muss gleich sein.
2. Die Hostnummern der Rechner müssen sich unterscheiden.
3. Die Hostnummer darf weder 0 noch 255 betragen.

Freie IP-Nummern

Für jede Klasse wurde in RFC1597 die Abkürzung für Request For Comment, übersetzt etwa: Bitte um Kommentare. Die RFC stellen so etwas wie die Norm des Internets dar. 1597 ein Bereich von Nummern definiert, der im Internet nicht weiter transportiert wird. Eine solche Nummer sollte für lokale Netze verwendet werden, sofern keine eigenen internetfähigen IP-Nummern reserviert worden sind. Internetfähige IP-Bereiche können von einem Internetprovider für die eigene Organisation reserviert werden. In Deutschland ist dafür das Europäische RIPE in Amsterdam zuständig (<http://www.ripe.org>), das hierfür aber einen ausführlichen Antrag mit Begründung des Bedarfs von heute bis in drei Jahren verlangt. Anders ausgedrückt heißt das, dass diese Nummern beliebig oft in der Welt verwendet werden können. Da solche Pakete im Internet nicht weitergeleitet werden, macht es gar nichts, wenn noch jemand anderes genau die gleichen Netzwerkadressen für sein Hausnetz verwendet.

[Die freien IP-Nummern]L|L Klasse & Nummernkreis
Class A & 10.0.0.0
Class B & 172.16.0.0 bis 172.31.0.0
Class C & 192.168.0.0 bis 192.168.255.0

Wenn eine kleine Firma für ihr internes TCP/IP-Netz IP-Nummern vergeben will, sollte sie sich unbedingt Adressen aus einem dieser Bereiche aussuchen. Da keine dieser Nummern im Internet vorkommt, gibt es auch keinen Web-, FTP- oder E-Mailserver mit einer solchen Adresse. Damit ist immer eindeutig, ob mit einer Adresse ein Rechner im lokalen Netz oder ein Server im Internet angesprochen wird. Es können auch durch eigene Adressen keine Internetadressen ausgeblendet werden. Ein zweiter Grund betrifft die Sicherheit gegen Angriffe aus dem Internet. Da diese Nummern im Internet nicht geroutet werden, kann ein Angreifer die Rechner des Netzes kaum direkt erreichen. Er kann ja die Adresse des Rechners nicht durch das Internet schleusen, da dieses nach RFC 1597 dort nicht transportiert wird.

Grundeinstellungen des Netzadapters: ifconfig

Mit dem Befehl `ifconfig` kann die Internet-Adresse jeder Schnittstelle festgelegt werden. Das funktioniert auch im laufenden Betrieb. Für das Umstellen der IP-Nummer ist also unter UNIX kein

Reboot erforderlich. Allerdings hängen an der IP-Nummer oft einige andere Konfigurationen. Der Aufruf von `ifconfig` hat folgende Struktur:

ifconfig Netzadapter IP-Nummer

`ifconfig` erwartet als ersten Parameter die Bezeichnung des Netzadapterdevices. Dabei wird allerdings der Pfadname `/dev` weggelassen. Die Namen dieser Gerätedateien unterscheiden sich je nach System.

[Namen für den ersten Netzwerkadapter]L|L System & Interfacename

SCO: & Je nach Hardware 3b0, wdn0 und so weiter

OS/2 ftp PCTCP: & nd0

Linux: & eth0

Solaris: & le0

FreeBSD: & ed0

Das einzige Device, das auf jeder Maschine den gleichen Namen hat, ist die Loopback-Einheit und heißt `lo0`. Dieses Pseudodevice verweist auf die eigene Maschine.

Soll mein Arbeitsplatzrechner auf die bereits genannten IP-Adresse umgestellt werden, lautet der Befehl unter Linux:

```
ifconfig eth0 192.168.109.144
```

Hier wird der ersten Ethernetkarte die Internet-Adresse 192.168.109.144 zugeordnet.

Der Befehl `ifconfig eth0` ohne weiteren Parameter zeigt die aktuelle Einstellung des Interfaces an. Beispiel:

```
gaston: # ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:00:E8:59:88:0F
          inet addr:192.168.109.144  Bcast:192.168.109.255
          Mask:255.255.255.0
          inet6 addr: fe80::200:e8ff:fe59:880f/10 Scope:Link
          inet6 addr: fe80::e859:880f/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:5 Base address:0xef40
```

Sie finden in dieser Meldung viele Details, die Sie nicht unbedingt verstehen müssen. Aber Sie werden sicher schon einiges wieder erkennen. In der ersten Zeile ist unter `HWaddr` (Hardware Address) die Ethernetadresse zu sehen. Sie sehen, dass die Broadcastadresse der Netzadresse entspricht und anstelle der Hostadresse 255 ist. Die Netzmaske ist erwartungsgemäß 255.255.255.0. Besonders interessant ist das Wörtchen `UP`. Es bedeutet, dass dieses Netzinterface aktiv ist. Sie können einen Netzadapter mit dem Befehl `ifconfig` abschalten. Sie müssen zum Abstellen den zusätzlichen Parameter `down` angeben. Zur Reaktivierung dient der Parameter `up`. Beispiel:

```
ifconfig eth0 down
```

Dies ist vor allem bei temporären Verbindungen von Bedeutung. Aber es kann auch im Servicefall ganz praktisch sein, eine Maschine schnell mal aus dem Netz zu nehmen.

Wenn es Schwierigkeiten mit der Netzanbindung gibt, kann ein Aufruf von `ifconfig` mit der Netzschnittstelle als Parameter anzeigen, ob die Schnittstelle aktiv ist. Wenn die Netzwerkhardware nicht korrekt erkannt wird, meldet der Befehl einen IOCTL-Fehler (I/O-Control). Das ist eine Fehlermeldung des Hardwaretreibers. Wie schon im Abschnitt über die Gerätedateien (siehe S. device) erwähnt wurde, ruft UNIX die Treiber mit dem Systemaufruf `ioctl()` auf, wenn es Informationen über das Gerät braucht.

Dauerhaftes Einstellen der IP-Nummer

Die IP-Nummer wird normalerweise in einer der rc-Dateien festgelegt. Dazu wird üblicherweise der Befehl `ifconfig` verwendet. Sie können die entsprechende Stelle leicht finden, indem Sie mit dem Befehl `grep` nach dem Wort »ifconfig« suchen. Allerdings wird die IP-Nummer unterschiedlich konfiguriert. Auf älteren UNIX-Systemen stand die IP-Nummer direkt hinter dem Aufruf von `ifconfig` und man hat sie dort im rc-Skript einfach geändert, wenn der Rechner auf Dauer eine andere Adresse haben sollte. Finden Sie an dieser Stelle eine IP-Nummer, können Sie sie bedenkenlos ändern. Steht hinter dem `ifconfig` eine Umgebungsvariable oder finden sich dort Kommandos zum Auslesen einer Datei, sollten Sie die eigentliche Quelle suchen und die IP-Nummer dort ändern. Wenn Sie an dieser Stelle eine IP-Nummer direkt angeben, funktioniert das natürlich auch. Falls aber jemand anders später versucht, die Nummer auf korrekte Art zu ändern, wird er sich wundern, warum nach jedem Reboot die alte IP-Nummer wieder aktiv ist.

Bei Solaris wird die IP-Adresse über den Hostnamen definiert. Für jeden Netzadapter gibt es eine Datei, die den Namen der Maschine oder besser des Adapters enthält. Beispielsweise steht in der Datei `/etc/hostname.le0` der Name `sol`. Damit erhält der Netzwerkadapter `le0` den Namen `sol`. Zur Bestimmung der IP-Nummer wird `sol` in der Datei `/etc/hosts` gesucht. Eine Änderung der IP-Nummer würde also in der Datei `/etc/hosts` erfolgen.

Unter Linux ist die Art der Bestimmung von der Distribution abhängig. SuSE verwendete dazu bis zur Version 8.0 seine Universalkonfigurationsdatei `/etc/rc.config`. Im Skript `/etc/init.d/network` finden Sie den `ifconfig`, der die Umgebungsvariable `IFCONFIG` benutzt, die in `rc.config` definiert ist.

Ab der Version 8.0 verwendet SuSE das Skript `/sbin/ifup`, um die Netzadapter zu initialisieren. Es wird vom rc-Skript `/etc/init.d/network` gestartet. Die bisher in der Datei `rc.config` gesammelten Konfigurationen werden nun im Verzeichnis `/etc/sysconfig` in mehrere Dateien und Unterverzeichnisse verteilt. Darin findet sich das Verzeichnis `network` und darin wiederum finden Sie mehrere Dateien, die die Konfiguration der verschiedenen Netzadapter bestimmen. Alle haben den Präfix `ifcfg-`. Die Datei für den Ethernetadapter heißt `ifcfg-eth0` und hat beispielsweise folgenden Inhalt:

```
BOOTPROTO=none
IPADDR=192.168.109.143
NETMASK=255.255.255.0
BROADCAST=192.168.109.255
NETWORK=192.168.0.0
STARTMODE=onboot
```

Die hier gesetzten Umgebungsvariablen werden in `/sbin/ifup` verwendet, um den Ethernetadapter zu initialisieren.

Gewisse Ähnlichkeiten zu der Konfiguration unter der Linuxdistribution von Red Hat sind unverkennbar. Ebenso wie dort sind im Verzeichnis `/etc/sysconfig` die Konfigurationsdateien zu finden. Bei Red Hat heißt das Unterverzeichnis `network-scripts`, in dem Sie schließlich die Datei namens `ifcfg-eth0` finden, die einen vergleichbaren Inhalt wie bei SuSE hat.

FreeBSD verwendet die Datei `rc.config` um eine Umgebungsvariable mit den Parametern des `ifconfig` zu setzen, die beim Booten in der Datei `rc.network` als Parameter für den Aufruf von `ifconfig` verwendet wird.

Auf einer unbekannten Maschine ist der Startpunkt der Suche nach der Stelle, wo die IP-Nummer festgelegt wird, der Ort, wo in den `rc`-Dateien der Befehl `ifconfig` abgesetzt wird. Dort müssen Sie ermitteln, woher `ifconfig` seine Informationen bekommt.

Die etwas einfachere Variante dürfte die Verwendung des jeweiligen Administrationstools sein, das auf allen Plattformen eine Möglichkeit zur Einstellung der IP-Nummern anbietet. Eine kurze Übersicht über diese Programme finden Sie ab Seite `admintools`. Die Tools haben den Vorteil, dass sie eine systemkonforme Einstellung der IP-Nummer gewährleisten.

« [TCP/IP, der Standard](#) | [TCP/IP, der Standard](#) | [Das Prüftool ping](#) »

Das Prüftool ping

Wenn es einen heimlichen Superstar unter den einfachen TCP/IP Werkzeugen für die Konfiguration von Netzen gibt, so ist es `ping`. Das Programm sendet kleine Pakete an den angegebenen Zielrechner. Da dieser ein Pingpaket sofort zurücksendet, kann Ping erstens feststellen, ob der Rechner gestartet ist, ob die Verkabelung korrekt ist und wie schnell und zuverlässig die Verbindung zu diesem Rechner ist.

Bevor Sie fremde Rechner anpingen, ist es kein schlechter Gedanke die eigene Nummer einmal per `ping` aufzurufen. Wenn das lokale TCP/IP nicht richtig konfiguriert ist, werden Sie auch keine Verbindung nach außen bekommen.

Bei einigen Implementationen wird nur die lapidare Meldung »xxx is alive« (beispielsweise SunOS) abgegeben. Andere Implementierungen (so beispielsweise MS-Windows) senden fünf Pakete und enden dann bereits mit einer Zusammenfassung. Beides ist unzureichend, wenn Sie beispielsweise sehen wollen, ob ein Wackelkontakt im Kabel ist. Durch entsprechende Optionen (-t) ist aber die typische Ausgabe von `ping` zu erhalten. Diese läuft durch, bis Sie `ctrl-C` drücken und zeigt dann eine Statistik an.

```
PING gaston.willemer.edu (192.168.109.144): 56 data bytes
64 bytes from 192.168.109.144: icmp_seq=0 ttl=255 time=0.172 ms
64 bytes from 192.168.109.144: icmp_seq=1 ttl=255 time=0.099 ms
64 bytes from 192.168.109.144: icmp_seq=2 ttl=255 time=0.095 ms
64 bytes from 192.168.109.144: icmp_seq=3 ttl=255 time=0.093 ms
64 bytes from 192.168.109.144: icmp_seq=4 ttl=255 time=0.094 ms
64 bytes from 192.168.109.144: icmp_seq=5 ttl=255 time=0.093 ms
64 bytes from 192.168.109.144: icmp_seq=6 ttl=255 time=0.093 ms
64 bytes from 192.168.109.144: icmp_seq=7 ttl=255 time=0.098 ms
-- gaston.willemer.edu ping statistics --
8 packets transmitted, 8 packets received, 0% packet loss
round-trip min/avg/max = 0.093/0.104/0.172 ms
```

Das Beispiel zeigt die Statistik einer gut laufenden Verbindung. Jedes zurückkommende Paket wird gemeldet. Die `icmp_seq` ist lückenlos. Bei dieser Nummer handelt es sich um Paketnummern. Da das Protokoll ICMP, unter dem `ping` läuft, verlorene Pakete nicht wiederholt, deutet eine durchgängige `icmp_seq` darauf hin, dass die Verbindung zuverlässig ist. Die Laufzeiten sind minimal. Dass das erste Paket etwas länger dauert, ist völlig normal.

Die Abkürzung `ttl` bedeutet `time to live`, also übersetzt etwa Lebensdauer. Ein Pingpaket versucht sein Ziel auch durch Wechsel des Netzes zu erreichen. Dabei passiert es jedesmal einen Router. Das ist ein Rechner, der die Verbindung zwischen zwei Netzwerken herstellt. Bei jedem Wechsel wird der `ttl`-Wert des Pakets um eins heruntergezählt. Sobald der Wert 0 ist, wird das Paket nicht mehr weitergeleitet. Damit wird verhindert, dass Pakete, die ihr Ziel nicht finden, endlos im Internet umherschwirren. Mit der Option `-t` kann der Startwert für `ttl` verändert werden.

Sie können die Paketgröße des `ping` beliebig festlegen. Dazu gibt es je nach System den Parameter `-s`. In manchen Fällen wird die Größe in Byte als weiterer Parameter hinter dem Ziel angegeben. Mit dieser Option können Sie große Datenpakete und damit eine große Netzlast erzeugen. Das folgende

Beispiel zeigt die Ausgabe von ping mit einer Paketgröße von 40000 Byte über eine nicht sehr verlässliche Verbindung.

```
gaston> ping -s 40000 192.168.109.137
PING 192.168.109.137 (192.168.109.137) from 192.168.109.144
: 40000(40028) bytes of data.
40008 bytes from 192.168.109.137: icmp_seq=6 ttl=255 time=73.689 msec
40008 bytes from 192.168.109.137: icmp_seq=13 ttl=255 time=73.742 msec
40008 bytes from 192.168.109.137: icmp_seq=21 ttl=255 time=73.624 msec
40008 bytes from 192.168.109.137: icmp_seq=23 ttl=255 time=72.995 msec
40008 bytes from 192.168.109.137: icmp_seq=38 ttl=255 time=73.151 msec
40008 bytes from 192.168.109.137: icmp_seq=39 ttl=255 time=72.456 msec
40008 bytes from 192.168.109.137: icmp_seq=40 ttl=255 time=72.279 msec
```

```
-- 192.168.109.137 ping statistics --
46 packets transmitted, 7 received, 84% loss, time 45090ms
rtt min/avg/max/mdev = 72.279/73.133/73.742/0.639 ms
gaston>
```

Der Wert für time ist recht konstant, aber natürlich höher als beim vorigen Versuch. Das liegt daran, dass die größeren Pakete länger unterwegs sind. An den heftigen Lücken bei icmp_seq ist aber erkennbar, dass immer wieder Pakete zerstört werden. Bei einem lokalen Netzwerk kann dies ein Hinweis auf einen Wackelkontakt, einen fehlenden Abschlusswiderstand oder einen defekten Hub sein.

« [Die TCP/IP-Nummer](#) | [TCP/IP, der Standard](#) | [Routing: Verbindung mehrerer Netzwerke](#) »

Routing: Verbindung mehrerer Netzwerke

Aus den verschiedensten Gründen werden Netzwerke in mehrere kleinere Netzwerke aufgeteilt. Aufgrund technischer Gegebenheiten ist dies notwendig, wenn beispielsweise unterschiedliche physikalische Netzträger verwendet werden, etwa Ethernet und Token-Ring. Aus geografischen Gründen ist es notwendig, wenn eine Filiale über Modem oder ISDN angekoppelt werden soll. Ein weiterer Grund kann das Ziel sein, die Netzlast in den einzelnen Netzen zu reduzieren.

Jeder Host kann von jedem Host im gleichen Netzwerk über seine Internetadresse erreicht werden. Dagegen ist eine Internetadresse eines fremden Netzes oder Subnetzes nur dann erreichbar, wenn die Pakete weitervermittelt werden. Dazu dient ein Gateway.

Unterabschnitte

- [Gateways](#)
- [Statische Festlegung einer Route](#)
- [Statisches Routing: Ein Beispiel](#)
 - [Gateway-Konfiguration zwischen Gintoft und Norgaardholz](#)
 - [Anbindung der Filialen](#)
 - [Von der IP zur Telefonnummer](#)
 - [Paketchen auf Reisen](#)
- [Subnetze](#)
- [Dynamisches Routen](#)
- [CIDR - Classless Inter-Domain Routing](#)

Gateways

Ein Gateway ist ein Computer mit zwei Netzwerkanschlüssen, die jeweils an ein anderes Netz angeschlossen sind. Jede Schnittstelle hat eine eigene Internet-Nummer, die dem entsprechenden Netzwerk zugeordnet ist. Kommt ein Paket auf der einen Netzwerkkarte für das jeweils andere Netz an, wird es vom Gateway in das andere Netz eingespeist.

Statische Festlegung einer Route

Jeder Rechner verwaltet eine Routing-Tabelle, in der er ablegt, auf welchen Wegen er welche Netze oder sogar einzelne Rechner erreichen kann. Um den Weg zu einem Rechner in einem fremden Netz zu definieren, wird der Befehl `route` verwendet. Abhängig vom Ziel gibt es drei Varianten, eine Route anzulegen:

`route add host Host gateway Gateway metric Metric`
`route add net Netz gateway Gateway metric Metric`
`route add default gateway Gateway metric Metric`

Der erste Parameter nach `route add` ist das Ziel der Route. Mit Gateway wird der weiterleitende Rechner angegeben, und mit Metric wird die Priorität der Route angegeben.

Der Parameter `metric` wird benötigt, wenn es mehrere Routen zu einem Ziel mit unterschiedlicher Geschwindigkeit gibt. Der schnellsten Route wird die höchste Priorität beispielsweise 1 vergeben. Fällt diese aus, kann dann auf die mit der niedrigeren Priorität 2 oder 3 ausgewichen werden. Die `metric`-Information wird nur beim dynamischen Routing verwendet, um eine Bewertung der Qualität der Strecke vorzunehmen. Bei statischem Routing ist der Parameter irrelevant.vgl. Hunt, Craig: TCP/IP Network Administration. O'Reilly, Sebastopol, 1994. p. 138.

Leider sind die Aufrufparameter des Kommandos `route` nicht ganz einheitlich. So kann bei manchen Versionen das Schlüsselwort `gateway` auch `gw` abgekürzt werden. Andere Implementationen brauchen die Schlüsselworte `gateway`, `metric`, `net` und `host` gar nicht, da sie sich aus der Reihenfolge bzw. aus der Art der Parameter von selbst ergeben. Der erste Parameter ist immer das Ziel. Ob das Ziel ein Host oder ein Netz ist, lässt sich direkt an der IP-Nummer ablesen. An zweiter Stelle steht immer das Gateway und der letzte Parameter ist immer die Metric.

Der Rechner 192.168.2.15 in Abbildung soll als Ausgangspunkt dienen, um die Einrichtung der Routingtabelle zu demonstrieren. Wollen Sie von dort eine Route auf den Rechner 192.168.3.32 legen, muss das Paket an das Gateway gesendet werden. Die vom Rechner aus erreichbare Adresse lautet 192.168.2.112. Der Befehl dazu lautet also:

```
route add host 192.168.3.32 gateway 192.168.2.112 metric 1
```

Sie brauchen nicht jeden einzelnen Rechner, sondern können mit einem Mal die Route für das gesamte Netz 192.168.3.0 angeben. Gateway und Metric bleiben gleich. Der Befehl lautet dann:

```
route add net 192.168.3.0 gateway 192.168.2.112 metric 1
```

Schließlich können Sie den Rechner anweisen, alle unbekannten Netzwerkadressen über ein bestimmtes Gateway hinauszuschleusen:

```
route add default gateway 192.168.2.212 metric 1
```

Der Eintrag `default` entspricht der IP-Nummer 0.0.0.0. Im Beispiel können Sie dann eine default-Route setzen, wenn es zu anderen Netzen kein weiteres Gateway gibt. In Netzen mit Internetzugang werden normalerweise alle direkt erreichbaren Netze mit expliziten Routen erreicht und das Gateway zum Internet auf `default` gesetzt.

Beim Anlegen einer Route können Sie auch die Netzmaske angeben. Das ist erforderlich, wenn im lokalen Netz Subnetze (siehe S. subnet) verwendet werden. Im Internet mit dem Einsatz von CIDR (siehe S. cidr) muss zu jeder Route die Netzmaske angegeben werden. Sie geben die Netzmaske durch einen zusätzlichen Parameter an, dem Sie das Schlüsselwort `netmask` voranstellen.

```
route add net 192.168.3.0 netmask 255.255.255.128
        gateway 192.168.2.112 metric 1
```

In diesem Fall würden die Adressen 192.168.3.1 bis 126 über das Gateway 192.168.2.112 geleitet. Der Backslash am Ende der Zeile bewirkt, dass die Befehlseingabe in der nächsten Zeile weitergeht.

Einträge in der Routing-Tabelle können wieder gelöscht werden:

route delete net Zieladresse Gateway
route delete host Zieladresse Gateway

Der Befehl `netstat` zeigt mit der Option `-r` die aktuellen Routingtabellen an.

```
gaston> netstat -r
Kernel IP routing table
Destination    Gateway        Genmask       Iface
192.168.109.0  *              255.255.255.0 eth0
loopback      *              255.0.0.0     lo
```

In der ersten Spalte stehen die Ziele, unter Gateway findet sich der Router, der aufgesucht wird, um das Ziel zu erreichen. Die Spalte Genmask beschreibt die Netzwerkmaske, die bei dem angegebenen Ziel vorausgesetzt wird und zu guter Letzt folgt das Interface, über das die Pakete abgesetzt werden.

« [Gateways](#) | [Routing: Verbindung mehrerer Netzwerke](#) | [Statisches Routing: Ein Beispiel](#) »

Unterabschnitte

- Gateway-Konfiguration zwischen Gintoft und Norgaardholz
- Anbindung der Filialen
- Von der IP zur Telefonnummer
- Paketchen auf Reisen

Statisches Routing: Ein Beispiel

Das Beispiel geht von einer Firma mit zwei Filialen und zwei Hauptgeschäftsstellen aus. Die eine Zentrale liegt in Gintoft, die andere in Norgaardholz. Jede der Filialen hat 100 PCs. Die Filialen liegen in Hamburg und Frankfurt/Oder und haben je 2 PCs.

Es werden die IP-Nummern für die Netze festgelegt:

[Übersicht über die IP-Nummern im Beispiel]L|L IP-Nummer & Bereich

192.168.108.0 & Gintofter Netz

192.168.109.0 & Norgaardholzer Netz

192.168.110.0 & Hamburger Netz

192.168.111.0 & Frankfurter Netz

Die Arbeitsplätze sollen von 1 bis 190 durchnummeriert werden. Die Server sollen als 201, weitere als 202, 203 und Router als 254 angesprochen werden.

Da Nummern sehr abstrakt sind, sollen die Rechner Namen erhalten. Die Arbeitsplätze in Gintoft erhalten den Praefix gin, die von Norgaardholz nor. In der Datei /etc/hosts (siehe S. hosts) hält UNIX die Liste, die die Namen auf die IP-Nummern abbildet. Die /etc/hosts sieht etwa so aus:

192.168.108.1	gin1
192.168.108.2	gin2
192.168.108.3	gin3
192.168.108.254	ginrout1
192.168.108.201	ginsrv1

192.168.109.1	nor1
192.168.109.2	nor2
192.168.109.3	nor3
192.168.109.254	norrout1
192.168.109.201	norsrv1

192.168.110.1	hh1
192.168.110.2	hh2
192.168.110.254	hhrou1

192.168.111.1	ffo1
---------------	------

```
192.168.111.2    ffo2
192.168.111.254  fforout1
```

Da es im Beispiel nur ein Gateway in jedem Netz gibt, ist die Konfiguration der Arbeitsplätze recht einfach: sie erhalten jeweils einen Default-Eintrag auf ihren Router. Für die Norgaardholzer Arbeitsplätze und den dortigen Server lautet er:

```
route add default norrout1 1
```

Bei Arbeitsplätzen mit MS Windows wird der Eintrag unter Systemkonfiguration - Netzwerk - Protokolle - TCP/IP - Gateway eingetragen. Hier ist Platz für ein oder mehrere default Routen. Ist es notwendig, einem Windowsrechner Routen zu verschiedenen Netzwerken über unterschiedliche Gateways einzutragen, erreichen Sie das nicht mehr über grafische Dialoge. Hier müssen Sie den route-Befehl in die AUTOEXEC.BAT schreiben. Wenn Sie im Beispiel davon ausgingen, dass Norgaardholz ein Router norrout2 den Zugang zum Internet realisierte, können Sie diesen in den Eintrag Gateway in der Systemkonfiguration eintragen. Für die Routen zu den Netzen Gintoft, Hamburg und Frankfurt müssten Sie folgende Befehle in der AUTOEXEC.BAT eintragen:

```
route add 192.168.108.0 mask 255.255.255.0 norrout1
route add 192.168.110.0 mask 255.255.255.0 norrout1
route add 192.168.111.0 mask 255.255.255.0 norrout1
```

Gateway-Konfiguration zwischen Gintoft und Norgaardholz

Die Konfiguration der Router ist interessanter. norrout1 hat zwei Interfaces. Das eine mit der Nummer 192.168.109.254 ist eine Ethernetkarte. Das andere Interface führt zur Telefonleitung und verbindet sich mit ginrout1. Zwischen ginrout1 und norrout1 gibt es also wiederum ein Netzwerk. Da dieses Netz nur den beiden Routern bekannt ist, verwenden Sie dort einfach irgendeine freie Nummer, beispielsweise 192.168.1.109 für norrout1 und 192.168.1.108 für ginrout1. Da dieses »Zwischennetz« nur zwischen den Routern bekannt ist, ist dieses nicht unbedingt erforderlich und wird von einigen Routern gar nicht benötigt. Allerdings wird das Verständnis des Routings durch den Wegfall auch nicht leichter. Die Routingtabellen des norrout1 lauten also:

```
# die Route auf das eigene, interne Netz
route add 192.168.109.0 192.168.109.254 1
# Gintoft über den Gintofter Router
route add 192.168.108.0 192.168.1.108 1
# Gintofter Router über WAN
route add 192.168.1.108 192.168.1.109 1
```

Damit kann norrout1 nur Nachrichten nach Gintoft bearbeiten. Andere Pakete fasst er nicht an. Ein Paket von Norgaardholz nach Gintoft würde nun seinen Weg finden, aber nicht mehr zurück, da in Gintoft der Router nicht weiß, wie das Paket nach Norgaardholz gesendet werden soll. Auf ginrout1 müssten die Routen also spiegelbildlich zum norrout1 eingerichtet werden. Noch einfacher wird es, wenn man davon ausgehen kann, dass norrout1 alle Verbindungen zu den Filialen übernimmt. Dann braucht ginrout1 nur den Eintrag

```
route add default 192.168.1.109 1
```

Anbindung der Filialen

Nun werden in Norgaardholz noch die Routen zu den Filialen in Hamburg und Frankfurt/Oder gebraucht. Wir vergeben der WAN Schnittstelle in Hamburg die Nummer 192.168.1.110 und der in Frankfurt 192.168.1.111. Durch die Routen

```
route add 192.168.110.0 192.168.1.110 1 # HH über HH Router
route add 192.168.111.0 192.168.1.111 1 # Ffo über Ffm Router
route add 192.168.1.110 192.168.1.109 1 # HH Router über WAN
route add 192.168.1.111 192.168.1.109 1 # Ffo Router über WAN
```

wird eine Verbindung von Norgaardholz nach Hamburg oder Frankfurt auf gleiche Weise geschlagen wie nach Gintoft. Natürlich muss auch hhrou1 und fforou1 mit den entsprechenden Routing-Einträgen versehen werden.

Was passiert aber, wenn Hamburg und Gintoft gleichzeitig arbeiten wollten? Damit jede Aussenstelle jederzeit eine freie Leitung vorfindet, werden drei Modems oder zwei ISDN-Karten besorgt. Da jede ISDN-Karte zwei B-Kanäle betreuen kann, reichen zwei ISDN-Anschlüsse, da jeder zwei B-Kanäle besitzt. Damit die Zuordnung klar ist, werden für den norrou1 mehrere WAN-Adapter eingerichtet.

```
route add 192.168.1.110 192.168.1.2 1 # HH Router über WAN 2
route add 192.168.1.111 192.168.1.3 1 # Ffm Router über WAN 3
```

Würde nun Gintoft deutlich häufiger mit Hamburg zu tun haben als Norgaardholz, würden Sie die WAN-Anbindung dorthin natürlich nicht über Norgaardholz, sondern über Gintoft legen. Dabei würden Sie sogar einen ISDN-Anschluss sparen, da Gintoft und Norgaardholz je zwei Kanäle brauchten. Allerdings würde das Routing geringfügig komplizierter, da der Router norrou1 für eine Verbindung nach Hamburg auf ginrou1 verweisen müsste.

Insbesondere bei Firmen mit vielen Filialen stellt sich die Frage, ob man nicht abwechselnd alle Filialen über eine Telefonleitung ansprechen kann. Das hängt natürlich davon ab, wie die Filialen mit den Rechnern in den Zentralen arbeiten. Findet nur nachts ein Datenabgleich statt, kann jede Filiale einzeln nacheinander aktualisiert werden. Dazu müssen die verschiedenen Netzadressen auf verschiedene Telefonnummern abgebildet werden. In dem Moment, wo aber die Filialen in Konkurrenz und zeitlich nicht vorhersehbar auf die Zentralen zugreifen und umgekehrt, empfiehlt sich eine Trennung nach Kanälen. Ein ISDN-Anschluss ist nicht sehr teuer und die Möglichkeit, dass eine Filiale die Zentrale nicht erreicht, weil die Leitung durch eine andere belegt ist, ist ausgeschlossen. Beim Thema Kosten sollte auch geprüft werden, ob es nicht sogar günstiger ist, eine Standleitung zu verwenden.

Von der IP zur Telefonnummer

Bisher wurde völlig übergangen, wie Sie eine TCP/IP-Verbindung über ISDN legen können. Tatsächlich ist das recht einfach möglich. Leider ist die Art der Konfiguration von dem verwendeten Router abhängig.

Als Beispiel soll ein Router unter Linux verwendet werden. Dort gibt es bei jeder Distribution das Paket `i4l` (ISDN for Linux). Die einzelnen B-Kanäle der ISDN-Karten werden auf ISDN-Devices abgebildet. Bei einer einfachen Karte lauten sie `isdn0` und `isdn1`. Für jedes Device wird mit dem Befehl `isdnctrl` die anzurufende Telefonnummer angegeben. Sobald das Interface angesprochen wird, wird die Verbindung gewählt. Werden eine gewisse Zeit keine Daten mehr transferiert, legt die Software auf,

um Verbindungskosten zu sparen.

Einen anrufbaren ISDN-Zugang wird jede Firma als Sicherheitsrisiko empfinden, solange nicht sichergestellt ist, dass er nur von der Filiale benutzt werden kann. Sie können für das Interface die Nummer festlegen, die der Anrufer haben muss. Alle anderen Telefonnummern werden dann durch das Interface abgewiesen. Es ist für einen Angreifer leicht, seine Telefonnummer zu unterdrücken. Aber das nützt ihm nichts, da er die Nummer der Filiale vorweisen muss und das ist nicht so einfach.

Eine weitere Sicherungsmöglichkeit besteht darin, einen Callback zu installieren. Die Filiale ruft an und signalisiert damit, dass sie eine Verbindung haben möchte. Die Zentrale nimmt den Anruf entgegen und stellt fest, welche IP-Nummer der Gegenüber hat. Sie weist den Anruf jedoch zunächst zurück. Nun wählt sie die hinterlegte Telefonnummer der empfangenen IP-Nummer und ruft so die Filiale sofort zurück. Auf diese Weise ist sicher, dass nur die Filiale einen Zugang hat. Selbst wenn es einem Angreifer gelingen würde, die Telefonnummer der Filiale zu fälschen, würde die Zentrale ihn anschließend nicht zurückrufen, sondern die richtige Filiale.

Paketchen auf Reisen

Zur Veranschaulichung des Routings soll ein Paket von gin3 nach norsrv1 und zurück gesendet werden. Es würde also auf gin3 der Befehl

```
ping norsrv1
```

abgesetzt. Zunächst würde über die Datei /etc/hosts von gin3 festgestellt, dass norsrv1 die IP-Nummer 192.168.109.201 hat. Das Paket erhält dies als Zieladresse und die eigene Adresse wird in den Absender gesteckt. Bereits dem Rechner gin3 ist klar, dass das Paket nicht zum eigenen Netz gehört. Gäbe es auf gin3 keinen Routing-Eintrag, erhielten Sie die Meldung:

```
no route to host
```

Durch die default-Route wird das Paket erst einmal zu ginrout1 geschafft. Dieser liest wiederum die Adresse. Wäre ihm keine Route bekannt, würde er das Paket mit der Fehlermeldung »no route to host« an gin3 zurückschicken. Da er aber eine Route hat, die an die ISDN Schnittstelle des norrout1 gerichtet ist, wird das Paket auf die Reise geschickt. norrout1 erkennt in der Adresse eine gültige Adresse für das Netz an seiner Ethernetkarte. Also reicht er das Paket hierhin weiter. Auf dem Ethernet findet es nun norsrv1.

Gemäß dem ping-Protokoll tauscht norsrv1 nun Sender- und Empfängeradresse und schickt das Paket wieder los. Würde hier nicht die default Route greifen, würde norsrv1 auf einem Paket sitzen, das nicht ins eigene Netz gehört und für das es keine Route gibt. Da eben gerade die Adresse von gin3 unbekannt ist, kann gin3 von diesem Problem nichts mitbekommen. Bleiben Pakete also ohne Fehlermeldung einfach aus, spricht alles dafür, dass sie korrekt das eigene Netz verlassen konnten, aber im fremden Netz nicht mehr den Weg zurück fanden. In diesem Fall könnten Sie sicher sein, dass das Problem auf norsrv1 oder norrout1 liegt.

Da es aber die default-Route gibt, geht das Paket an norrout1. norrout1 erkennt die Nummer als Netznummer von Gintoft und besitzt eine Route dorthin über ginrout1. Dieser gibt das Paket auf das Ethernet, so dass es zu gin3 zurück findet.

Subnetze

Anstatt für jedes Teilnetz eine eigene Netzkennung zu verwenden, können Sie auch ein großes Netz in Unternetze zerteilen. Von außen ist diese Unterteilung nicht zu sehen und das Netz wird von fremden Routern als ein einheitliches Netz angesehen. Erst wenn Pakete ins Innere des Netzes gelangen, wird es durch die internen Router weitergeleitet.

Um ein Netzwerk derart zu unterteilen, müssen Sie die Netzmaske verändern. Die Netzmaske gibt an, welcher Teil der IP-Nummer das Teilnetz bestimmt und welcher Teil der Nummer einen einzelnen Rechner festlegt. Die Maske enthält für jedes Bit, das zur Netzwerkadresse gehören soll, eine 1. Für die drei Klassen von Netzen sind folgende Netzmasken Standard:

[Standardnetzmaske]L|L|L Klasse & hexadezimal & dezimal

Class A & 0xFF000000 & 255.0.0.0

Class B & 0xFFFF0000 & 255.255.0.0

Class C & 0xFFFF0000 & 255.255.255.0

In der folgenden Tabelle werden drei IP-Nummern der verschiedenen Netzklasse anhand der Standardnetzmaske in Netz- und Hostanteil aufgegliedert.

[gpAbbildung8078]

Die Netzwerkmaske ist bitweise konfigurierbar. Und zwar kann der Anteil der Netzkennung vergrößert werden. Wird eine Netzwerkmaske um ein Bit erhöht, wird damit das Netz in zwei Teilnetze zerlegt. Bei einem Class C Netz würde dann die Netzmaske 255.255.255.128 lauten. Die 128 mag etwas überraschen. Aber von dem rechten Byte wird das am weitesten links stehende Bit verwendet. Die Dualdarstellung von 128 ist 1000000. Damit gehören alle Hostnummern, die größer als 128 sind zu dem einen Teilnetz und alle die kleiner sind, zu dem anderen Teilnetz.

Da die Netzmaske immer so aufgebaut ist, dass in der Dualdarstellung von links beginnend nur Einsen und ab einer gewissen Grenze dann nur noch Nullen folgen, gibt es eine alternative Schreibweise, die hinter einem Schrägstrich nur angibt, wieviele Einsen die Netzmaske enthält. Für ein einfaches Class C Netzwerk sieht das so aus: 192.168.109.144/24, bei der obigen Netzmaske 255.255.255.128 käme noch ein Bit hinzu, also 192.168.109.144/25. Man spricht auch von der CIDR Schreibweise (siehe S. cidr).

Es gehen bei der Teilung in Subnetze auch IP-Nummern verloren, die nicht mehr verwendet werden können. Denn für beide Teilnetze gilt die Regel, dass Adressen, bei denen alle Hostbits 0 oder 1 sind, nicht für die Adressierung von Rechnern verwendet werden dürfen. Das wären im Beispiel die Nummern 0, 127, 128 und 255.

In der folgenden Grafik ist eine Class B Adresse mit Subnetzmaske dargestellt. Die Standardnetzmaske ist bei Class B 255.255.0.0. In diesem Fall soll das erste Halbbyte der Hostkennung noch mit zur Netzadresse genommen werden. Von links beginnend haben die Bits den Wert 128, 64, 32 und 16. Die andere Hälfte des Bytes bleibt Null, da es nicht in die Netzmaske

einfließen soll. Die Summe aus 128, 64, 32 und 16 ist 240. Das ist damit die Netzmaske des dritten Bytes.

[gpAbbildung8108]

Eine Aufteilung des Netzes kann die Netzbelastung reduzieren. Gehen wir von einer Softwarefirma aus, die das Class C Netz mit der Nummer 192.168.2.x hat. In dem unteren Stockwerk befindet sich die Verwaltung, die auf einer Netzwerkplatte in unregelmäßigen Abständen Dokumente abstellen und in der Kundendatei suchen. Im oberen Stockwerk sitzen die Programmierer, die Netzwerkprogramme schreiben und von Zeit zu Zeit Belastungstests machen. Immer wenn solche Tests anlaufen, kommt die Sekretärin nicht mehr an ihre Kundendaten heran, weil das Netz überlastet ist. Schafft sie es trotzdem, werden durch ihre zusätzlichen Zugriffe die Testergebnisse verfälscht.

Man teilt das Netz logisch durch die Netzmaske. Physisch werden die Kabel getrennt und durch ein Gateway verbunden, das zwei Netzwerkkarten besitzt. So kann ein Programmierer seine Dokumentation zur sprachlichen Kontrolle immer noch an die Sekretärin senden. Das Gateway wird die Pakete von dem einen Netz ins andere übertragen. Der restliche Netzverkehr bleibt im jeweiligen Teilnetz.

Die Netzaufteilung durch Subnetting ist nur für Rechner im Netz sichtbar, da die Maske auf den lokalen Rechnern im Netzdevice und in den Routingtabellen festgelegt wird. Es ist also wichtig, dass alle Rechner die gleiche Netzmaske bekommen. Für außenstehende Rechner erscheint das Netz homogen, da sie eine Subnetzmaske für ein fremdes Netz nicht kennen und so die Standardnetzmaske vermuten müssen. Unter CIDR sind die Netzmasken auch außerhalb des lokalen Netzes bekannt. Hier werden die Netzmasken von den Routern verwaltet und weitergegeben. Aber auch hier gilt, dass ein lokales Subnetting nicht unbedingt mit der nach außen bekannten Netzmaske übereinstimmen muss.

« [Statisches Routing: Ein Beispiel](#) | [Routing: Verbindung mehrerer Netzwerke](#) | [Dynamisches Routen](#) »

Dynamisches Routen

Insbesondere das Internet mit seinen vielen Teilnetzen und der ständigen Veränderung erfordert ein Routingverfahren, das sich dynamisch anpassen kann. Die Routingtabellen werden dabei nicht vom Administrator festgelegt, sondern von Dämonen verwaltet, die mit denen der Nachbarnetze Informationen über die Qualität der Verbindung und über die Erreichbarkeit anderer Netze austauschen. Auf diese Methode werden nicht nur Engpässe oder gar Ausfälle von Leitungen erkannt. Es ist sogar möglich, die Pakete über die nächstbeste Verbindung zu schicken. In den Routingtabellen erhält der Parameter metric eine zentrale Bedeutung, weil er bei schlechter Verbindung erhöht wird.

In einem Firmennetz werden diese dynamischen Verfahren normalerweise nicht eingesetzt. Dort werden üblicherweise keine Ausfalleleitungen gelegt, auf die ein dynamisches Verfahren ausweichen könnte. Dort wo kritische Verbindungen durch Ersatzleitungen abgesichert werden, handelt es sich um ein einfaches Backup, das Sie schnell durch das manuelle Ändern von zwei Routingeinträgen in Betrieb nehmen können. Die Komplexität ist überschaubar und die Veränderungen in den Netzen sind vorhersehbar und meist gut geplant.

Dynamisches Routen wird beispielsweise durch den Dämon `routed` realisiert, der das RIP (Routing Information Protocol) implementiert. Der Dämon `gated` beherrscht neben RIP auch das externe Routing EGP (Exterior Gateway Protocol). Dynamisches Routen ist das Rückgrat des Internets. Da die Router im ständigen Austausch über die Qualität der Leitungen stehen, können defekte Leitungen durch Anpassung der Routingtabellen automatisch umgangen werden.

Das EGP informiert über die Erreichbarkeit autonomer Systeme. Ein autonomes System kann ein komplexes Netzwerk mit diversen internen Routern sein. Es muss nur nach außen abgeschlossen sein. vgl. Hunt, Craig: TCP/IP Network Administration. O'Reilly, Sebastopol, 1994. pp. 142

CIDR - Classless Inter-Domain Routing

Mit zunehmender Beliebtheit des Internets wurden dessen Engpässe immer sichtbarer. So war die Idee, 32 Bit für die IP-Nummer zu verwenden, nicht weitreichend genug, wie sich bei Ausbau des Netzes herausstellte. Zwar kann man mit vier Byte etwa 4 Milliarden Rechner durchnummerieren, und das war immerhin die damalige Größe der Weltbevölkerung, aber bei genauerem Hinsehen war die Zahl doch nicht so großzügig. So gehen bei jedem Netz zwei Adressen für die 0 und die Broadcastadresse verloren. Dazu kommt, dass eine Firma, die ein Class C Netzwerk betreibt, üblicherweise nicht alle 254 Adressen auch wirklich einsetzt.

So wurde um 1993 das CIDR eingeführt.vgl. Nemeth, Evi / Snyder, Garth / Seebass, Scott / Hein, Trent R.: UNIX Systemverwaltung. Markt+Technik - Prentice Hall, 2001. S. 357-360. Die Idee war, dass man jeder Netzwerkadresse eine Netzmaske mitgab und erst durch diese bestimmt wird, wieviele Rechner in ein Teilnetz gehören. Wenn es nun noch gelänge, die benachbarten IP-Nummern lokal zu bündeln, könnte man sogar die Anzahl der Routingeinträge reduzieren. CIDR war eigentlich als Übergangslösung bis zum Einsatz der neuen 128 Bit TCP/IP-Nummern des IPv6 (siehe S. ipv6) gedacht.

Die Änderungen durch die neue Norm konnten recht problemlos und schnell im Internet umgesetzt werden, da sie nur die Routingtabellen betreffen. Oft müssen sie nur in den Routern gesetzt werden, da die Arbeitsplätze in den meisten Fällen ohnehin nur die Defaultroute zum Gateway verwenden. Da Router ihre Routingtabellen dynamisch austauschen, ist die Konsistenz leicht zu gewährleisten. CIDR war neben dem Einsatz von Masquerading (siehe S. masquerading) die entscheidende Technik, um das Problem der ausgehenden IP-Nummern zu umgehen.

In lokalen Netzwerken ist der Einsatz von CIDR nicht besonders sinnvoll, da hier keine dynamischen Routingverfahren zur Verfügung stehen und damit die Netzmasken von Hand konsistent gehalten werden müssten. Verwendet man im lokalen Netzwerk dagegen die normalen Netzwerkklassen, braucht man auf den einzelnen Arbeitsplätzen keine Netzmasken zu pflegen.

Namensauflösung

Während Computer mit Zahlen wunderbar zurecht kommen, verwenden Menschen lieber Namen. Das Auflösen der Namen in Zahlen und umgekehrt kann aber wieder dem Computer übertragen werden, denn im Netzwerk ist es wichtig, dass Namen konsistent bleiben.

Unterabschnitte

- [Der Host- und Domainname](#)
 - [Die Datei /etc/hosts](#)
 - [Die Datei /etc/services](#)
 - [Domain Name Service: DNS](#)
 - [DNS-Client](#)
 - [DNS-Server named](#)
 - [Testen](#)
 - [Root-Cache Dateien](#)
 - [Änderungen zu BIND 8 und 9](#)
 - [Network Information Service: NIS](#)
 - [Aufbereiten der Dateien](#)
 - [NIS-Server starten](#)
 - [Starten eines NIS-Clients](#)
 - [Netzgruppen: /etc/netgroup](#)
-

Der Host- und Domainname

Jede UNIX-Maschine hat standardmäßig einen Namen, den so genannten Hostnamen. Dieser erhält aber erst seine besondere Bedeutung, wenn die Maschine ins Netz geht, da sie dann unter diesem Namen angesprochen werden kann. Die Zuordnung des Hostnamens für die Maschine erfolgt durch den Befehl

hostname Name

Dieser Befehl wird im Allgemeinen in einer der rc-Dateien beim Hochfahren des Systems ausgeführt. Seinen Parameter entnimmt der Befehl `hostname` oft einer besonderen Datei, wie bei Linux der der Datei `namens /etc/HOSTNAME`. Solaris besitzt sogar für jeden Netzadapter eine eigene Namensdatei, beispielsweise `/etc/hostname.le0`.

Im Internet wird der einzelne Computer vollständig mit seinem Namen und seiner Domäne identifiziert. Normalerweise wird die Domäne durch einen Punkt abgetrennt an den Hostnamen angehängt. Beispielsweise besagt der Name `gaston.willemer.edu`, dass der Hostname `gaston` lautet und der Rechner zur Domäne `willemer.edu` gehört. Innerhalb seiner Domäne braucht der Rechner nicht mit seinem Domänennamen angesprochen zu werden. Diese dreigeteilten Namen kennen Sie aus dem Internet. So ist beispielsweise bei `www.willemer.de` der Anteil `www` der Name des Rechners in der Domäne `willemer.de`.

Um genau zu sein, baut sich ein Domänenname von hinten nach vorn auf. So bezeichnet man das »de« als Toplevel Domain. Neben `de` für Deutschland gibt es unter anderem `nl` für Niederlande oder `dk` für Dänemark. Die Domänen in den USA wurden nicht in einer nationalen Toplevel Domain zusammengestellt, sondern werden in `com` für kommerzielle Organisationen, `gov` für Regierungsstellen und `edu` für Universitäten, Bildungs- und Forschungseinrichtungen eingeteilt. Von der Toplevel Domain wird mit dem Punkt die eigentliche Domäne abgeteilt. Diese Domänen können noch einmal in beliebig viele Subdomänen unterteilt werden, jeweils durch einen Punkt. Wird mit dem Namen ein Computer bezeichnet, ist nur der erste Begriff bis zum ersten Punkt der Hostname. Beispielsweise könnte der Praktikumsrechner des Fachbereichs Informatik an der Universität Gintoft so heißen:

`praktikum.informatik.universitaet.gintoft.de`

Die Datei /etc/hosts

Um einen fremden Rechner nicht immer über seine Internet-Nummer ansprechen zu müssen, gibt es die Datei /etc/hosts, die jeder Internet-Nummer einen oder mehrere Namen zuordnet. Die Struktur eines Eintrags in der Datei /etc/hosts sieht folgendermaßen aus:

ipadresse name nickname ...# Kommentar

Links steht immer die IP-Nummer des Rechners. Das ist die bereits bekannte »dotted decimal« Schreibweise, also vier Dezimalzahlen durch Punkte getrennt. Es folgen ein oder mehrere Namen. Der erste Name ist der wichtigste. Wird eine Verbindung von außen aufgebaut, erfährt die Maschine nur die Internetnummer. Um diesem Rechner einen Namen zu geben, wird der erste Name aus der /etc/hosts verwendet. Auch bei der Zuordnung von Rechten an bestimmte Rechner wird meist ein Name in die jeweilige Konfigurationsdatei eingetragen. Will ein Rechner seine Rechte geltend machen, erfährt das System seine IP-Nummer. Diese wird dann durch den ersten Namen in der hosts-Datei ersetzt und mit dem eingetragenen Rechnernamen verglichen. Der erste Name ist also der »offizielle«, die anderen bezeichnet man als nickname (Spitzname). Nach dem Hashzeichen (#) können Sie Kommentare einfügen. Auch der Hostname der eigenen Maschine sollte in der Datei eingetragen werden, da sonst Dienstanfragen an den eigenen Host über das TCP/IP nicht erkannt werden. Bei Solaris ist es sogar unverzichtbar, da sonst die Maschine ihre IP-Nummer nicht findet.

Ein Beispiel für eine /etc/hosts wurde bereits im Routingbeispiel auf Seite hostsbeispiel gezeigt.

Bei der Namensauflösung wird normalerweise zunächst in der Datei /etc/hosts nachgesehen. Um genau zu sein, wird die Reihenfolge in der Datei /etc/host.conf festgelegt (siehe DNS; S. dns). Das bedeutet, dass man Namen, die durch einen zentralen Server festgelegt werden sollen, hier nicht eintragen sollte. Würde beispielsweise die /etc/hosts folgendermaßen aussehen, dann könnte der Webserver www.galileo.de nicht mehr erreicht werden, weil er der lokalen IP-Adresse von gaston zugeordnet wäre. Das bräuchte Sie nicht zu bekümmern, denn der Verlag mit den fantastischen Büchern hat die Webadresse www.galileocomputing.de.

127.0.0.1	localhost	
192.168.109.144	gaston.willemer.edu	gaston
192.168.109.144	www.galileo.de	

Sie machen sich übrigens keine Freunde im TCP/IP-Umfeld, wenn Sie Ihre Hostnamen in Großbuchstaben setzen. In den meisten Fällen unterscheiden die Netzprotokolle zwischen Groß- und Kleinschreibung, so dass später jeder, der mit diesen Rechnern zu tun hat, mit den Großbuchstaben hantieren muss.

Die Datei /etc/services

Eine Netzwerkanfrage besteht nicht nur aus Kenntnis des Rechners. Wenn eine Anfrage an einen Netzdienst wie HTTP, FTP oder der Druckdienst gestellt wird, muss man den entsprechenden Prozess wie `httpd`, `ftpd` oder `lpd` auf diesem Rechner erreichen, der die Anfrage beantwortet.

Um eine Verbindung zwischen zwei Prozessen aufzubauen, wird auf jeder Maschine ein Socket (engl. Steckdose) verwendet. Die Sockets eines Rechners sind durchnummeriert. Der Client braucht einen Socket, um über diesen später eine Antwort zu bekommen. Dazu bekommt er irgendeine Socketnummer zugeteilt, die gerade frei ist. Um einen bestimmten Dienst und damit dessen Server zu erreichen, muss der Client den Socket kontaktieren, der von dem Serverprozess betreut wird.

Für Standarddienste hat man die Nummern der Sockets festgelegt. Sie erreichen den Webserver eines Rechners im Regelfall über den Socket 80. Das Protokoll heißt HTTP (Hypertext Transfer Protocol) und der Serverprozess, der Anfragen über dieses Protokoll beantwortet heißt `httpd`. In Abbildung sehen Sie zwei Clientprozesse namens `mozilla` und `netscape`, die auf dem Rechner 192.168.109.137 laufen. Sie stellen beide Anfragen an den Webserver auf dem Rechner 192.168.109.144. Die Sockets, die die Clients verwenden, haben eine beliebige Nummer, die ihnen zufällig vom System zugeordnet werden. Sie rufen den Socket 80 auf dem Zielrechner. Diese Nummer hat der Webserverprozess `httpd` beim Start angefordert.

Im Zusammenhang mit den festgelegten Serversockets wird von einem Port gesprochen. Um den Nummern Namen zuzuordnen, gibt es die Datei `/etc/services`. Hier sehen Sie einen Ausschnitt:

<code>ftp-data</code>	<code>20/tcp</code>	<code># File Transfer [Default Data]</code>
<code>ftp-data</code>	<code>20/udp</code>	<code># File Transfer [Default Data]</code>
<code>ftp</code>	<code>21/tcp</code>	<code># File Transfer [Control]</code>
<code>telnet</code>	<code>23/tcp</code>	<code># Telnet</code>
<code>telnet</code>	<code>23/udp</code>	<code># Telnet</code>
<code>http</code>	<code>80/tcp</code>	<code># world wide web HTTP</code>
<code>http</code>	<code>80/udp</code>	<code># world wide web HTTP</code>
<code>www</code>	<code>80/tcp</code>	<code># world wide web HTTP</code>
<code>www</code>	<code>80/udp</code>	<code># world wide web HTTP</code>
<code>www-http</code>	<code>80/tcp</code>	<code># world wide web HTTP</code>
<code>www-http</code>	<code>80/udp</code>	<code># world wide web HTTP</code>

Unter der Nummer 80 finden Sie den Dienst `http` und unter 21 den Dienst `ftp`. Und da `ftp` in der Regel unter der Portnummer 21 zu erreichen ist, spricht man von einem well known port (übersetzt etwa: wohlbekannter Hafen).

Das `tcp` hinter dem Schrägstrich ist die Kennung für das Protokoll. Neben TCP findet sich in der `/etc/services` noch das Protokoll `udp`. Es kann für dieselbe Nummer beide Protokolle nebeneinander geben. Ein weiteres Protokoll wurde im Zusammenhang mit dem Befehl `ping` bereits vorgestellt: ICMP.

TCP (Transmission Control Protocol) hat die Aufgabe, Daten sicher durch das Netzwerk zu transportieren. Dabei überprüft es die Netzpakete auf ihre Unversehrtheit anhand einer Prüfnummer.

Beschädigte Pakete werden neu angefordert. Geraten Pakete in der Reihenfolge durcheinander, weil sie beispielsweise unterschiedliche Wege genommen haben, oder weil defekte Pakete neu angefordert wurden, sorgt TCP für die korrekte Reihenfolge. Den Anwendungsprogrammen gegenüber stellt es einen Datenstrom zur Verfügung. Das Programm muss die Aufteilung auf Pakete also nicht selbst durchführen. Zu guter Letzt veranlasst TCP den geregelten Verbindungsabbau.

UDP (User Datagram Protocol) ist da sehr viel einfacher. Es wird nicht eine Verbindung aufgebaut, sondern es werden einfach Pakete versendet. UDP prüft nicht den Empfang oder die Reihenfolge der Pakete. Es wird lediglich gewährleistet, dass die Pakete unversehrt sind. Dieser fehlende Komfort kommt der Geschwindigkeit zu gute. So arbeitet beispielsweise das verteilte Dateisystem NFS auf Basis von UDP.

ICMP (Internet Control Message Protocol) ist noch einfacher. Es transportiert keine Daten, sondern Fehlermeldungen und Diagnoseinformationen. Das Protokoll wird meist unsichtbar von den Netzwerkschichten verwendet. Lediglich durch den Befehl `ping` hat der Anwender mit diesem Protokoll direkt zu tun.

« [Die Datei /etc/hosts](#) | [Namensauflösung](#) | [Domain Name Service: DNS](#) »

Unterabschnitte

- [DNS-Client](#)
 - [DNS-Server named](#)
 - [Testen](#)
 - [Root-Cache Dateien](#)
 - [Änderungen zu BIND 8 und 9](#)
-

Domain Name Service: DNS

Der Domain Name Service dient der Auflösung von Hostnamen zu Internetnummern und umgekehrt und übernimmt damit die Aufgabe, die die Datei /etc/hosts lokal erfüllt. Der Vorteil des DNS liegt auf der Hand: Änderungen müssen nur an einer Stelle durchgeführt werden und gelten für alle Rechner im Netz.

Die Bedeutung von DNS reicht aber weit über das lokale Netzwerk hinaus. Der Dreh- und Angelpunkt für einen Zugriff auf das Internet ist die Anbindung an einen DNS-Server. Sobald diese Anbindung erfolgt ist, erhalten Namen wie `www.apple.de` oder `arnold@willemer.de` erst die Zuordnung zu den richtigen IP-Nummern.

Eine Domäne ist nicht unbedingt deckungsgleich mit einem physikalischen Netzwerk, sondern bezeichnet den Bereich, für den der Namensserver eingesetzt wird. Die kleine Firma, die als Beispiel für das Routing diente, hat zwar mehrere TCP/IP-Netze, wird aber sicher nur eine Domäne einrichten.

BIND (Berkeley Internet Name Domain) ist die wichtigste Implementation des DNS. Die Konfiguration des Clients erfolgt in `resolv.conf` und `host.conf`. Beide Dateien befinden sich im Verzeichnis `/etc`. Der Server verwendet zwei weitere Dateien je Domäne zur Namens- bzw. zur Nummernauflösung.

Als Primary Server bezeichnet man denjenigen Namensserver, der die Autorität für die Domäne hat. Um den Ausfall des Primary Servers abzusichern, stellt man einen Secondary Server ins Netz. Dieser übernimmt die Namensauflösung, wenn der Primary Server ausfällt. Seine Informationen übernimmt der Secondary Server übers Netz vom Primary Server. Damit muss nur der Primary Server aktualisiert werden, wenn beispielsweise neue Rechner hinzukommen.

Als weitere Variante gibt es den Cache Only Server. Cache Only Server kopieren sich die Informationen des Primary Servers. Sie werden beispielsweise in Netzen verwendet, die zur Namensauflösung ansonsten eine Wählleitung in Anspruch nehmen müssten.

DNS-Client

Der Client wird als Resolver (engl. resolve: auflösen) bezeichnet. Dabei ist der Resolver nicht ein eigenständiger Prozess, sondern ist mittels Funktionsbibliothek quasi in den einzelnen Programmen enthalten. Alle Programme, die mit dem Namen von Rechnern arbeiten, verwenden den Systemaufruf `getservbyname()`, um die IP-Nummer dieses Rechners zu ermitteln (siehe S. `getservbyname`).

Sobald UNIX diesen Aufruf erhält, prüft `getservbyname()` anhand der Einträge in der Datei `host.conf`, ob die Namensauflösung zuerst per `hosts` oder per `DNS` erfolgt. Soll der Name per `DNS` ermittelt werden, wird in der `resolv.conf` nachgesehen, welche `DNS-Server` zuständig sind. Dann wird direkt vom Programm auf der `DNS-Server` kontaktiert. Alle drei Dateien befinden sich im Verzeichnis `/etc`.

In der Datei `host.conf` legt die Zeile, die mit `order` beginnt, fest, in welcher Reihenfolge die Namensauflösungsverfahren aufgerufen werden. Im unteren Beispiel wird zuerst in der `/etc/hosts` nachgesehen und anschließend `BIND` verwendet. Diese Reihenfolge ist normalerweise auch sinnvoll, da der Zugriff auf die lokalen Datei effizienter als die Suche im Netz ist.

```
order hosts bind
multi on
```

In der Datei `/etc/resolv.conf` wird definiert, zu welcher Domäne der Rechner gehört und welche Rechner Namensserver sind. Um meinen Arbeitsplatzrechner an den `DNS-Dienst` des Internet zu koppeln, reicht eine kleine `resolv.conf`. Beispiel:

```
domain      willemer.edu
nameserver  194.25.2.129      # frage den ISP
```

Eine Namensanfrage wird aufgrund der `host.conf` zunächst lokal in der `/etc/hosts` bedient und alle anderen Namen würden beim `DNS` des Providers gesucht.

Wollen Sie das `TCP/IP-Netz` der Beispielfirma an das Internet anbinden, reicht es, eine etwas größere `resolv.conf` zu verwenden. Dabei wird hier angenommen, die Firma hätte die Domäne `firma.de`.

```
domain      firma.de
nameserver  192.168.108.201 # frage zuerst unseren Server
nameserver  194.25.2.129    # dann frage den ISP
```

Zuerst werden die Namen lokal geprüft. Damit gehen lokale Namensauflösungen nicht ins Internet. Erst wenn der Name hier unbekannt ist, wird der `DNS-Server` des Internet Service Providers (`ISP`) angefragt.

Zu einer vollständigen Anbindung ist natürlich ein geeignetes Routing einzurichten. Da die meisten Firmen über keine eigenen internetfähigen `IP-Adressen` verfügen, muss noch ein `Proxy` (siehe S. proxy) oder ein `Masquerading` (siehe S. masquerading) eingerichtet werden. Beides sind unterschiedliche Mechanismen, um über einen anderen Rechner, der eine Verbindung zum Internet hat, einen Zugang zu erhalten, obwohl der eigene Computer keine internetfähige Adresse hat.

DNS-Server named

Der Serverprozess des `DNS` heißt `named`. Er verwendet mehrere Dateien zur Konfiguration. Die erste Datei ist die `/etc/named.boot`, die die Rolle des Rechners im `DNS` festlegt und die Namen und Orte der weiteren Dateien definiert. Dann gibt es für jede Domäne zwei weitere Dateien. Die eine bildet die Namen auf `IP-Nummern` um und die andere nennt den Standardnamen für eine `IP-Nummer`.

Ein `Primary Server` hat beispielsweise folgende Einträge in der Konfigurationsdatei `named.boot`:

```
primary willemer.edu /var/named/name2nr
primary 109.168.192.IN-ADDR.ARPA /var/named/nr2name
cache . /var/named/named.root
forwarders 194.25.2.129
```

Primary Server wird ein DNS-Server genannt, der die Namen für die Domäne verantwortlich verwaltet. Die erste Zeile drückt aus, dass dieser Rechner ein Primary Server für die Domäne willemer.edu ist. Die Namensinformationen der Domäne werden in der Datei /var/named/name2nr gepflegt. Für den umgekehrten Weg, also aus den Nummern die Namen zu ermitteln, wird nr2name verwendet. Die Cachedatei dient als Basis für das Vorhalten der Namen aus dem Internet. Wie Sie an diese Datei gelangen, wird später beschrieben. Der letzte Eintrag gibt den Namensserver an, an den die Anfrage weitergeleitet werden soll, wenn der Name aus einer fremden Domäne stammt. Beispielsweise kann hier der DNS-Server des Internet Providers stehen. Ein Secondary Server hätte eine ganz ähnliche /etc/named.boot:

```
directory /var/named
secondary willemer.edu 192.168.109.144 name2nr
secondary 109.168.192.IN-ADDR.ARPA 192.168.109.144 nr2name
```

Der Unterschied liegt in der ersten Spalte der zweiten und dritten Zeile. Sie besagt, dass dieser Rechner ein Secondary Server für willemer.edu ist. Sie gibt ferner die IP-Nummer des Primary Servers an. Schließlich wird definiert, in welcher lokalen Datei die Informationen vom Primary Server abgelegt werden. Die erste Zeile (directory) ist lediglich eine Abkürzung, damit man den Pfadnamen nicht bei jeder Dateiangabe wiederholen muss.

In diesem Beispiel wurde die Datei name2nr in der /etc/named.boot als Namensdatei angegeben. In dieser Datei dient das Semikolon als Kommentarzeichen.

```
@      IN SOA      mail.willemer.edu.    root.mail.willemer.edu.
      (
      200111303      ;serial
      360000         ;refresh every 100 hours
      3600           ;retry after 1 hour
      3600000        ;expire after 1000 hours
      360000         ;default ttl is 100 hours
      )
;      Wer ist der zuständige Namensserver
      IN NS gaston.willemer.edu.

;      wer sind die zuständigen Mailserver
      IN MX 10 mail.willemer.edu.

localhost      IN A      127.0.0.1

;      Alle Rechner in der eigenen Domain
mail           IN A      192.168.109.137
asterix        IN CNAME mail
gaston         IN A      192.168.109.144
powermac       IN A      192.168.109.141
```

Die erste Zeile beginnt mit einem @. Es steht für den lokalen Domännennamen. Hier könnte also auch willemer.edu stehen. Die Abkürzung SOA bedeutet Start Of zone Authority. Die so bezeichnete Zeile gibt den Standardnameserver der Domäne und die E-Mailadresse des Verantwortlichen an. Allerdings

wird das @ in der Mailadresse durch einen Punkt ersetzt. gaston ist der Namensserver (IN NS). Mailserver der Domäne ist der Rechner mail (IN MX), der aber auch unter dem Namen asterix im Netz agiert.

In der Klammer werden Parameter gesetzt, die Informationen über das Update-Verhalten festlegen. Die Zeile mit dem Kommentar serial kann irgend eine Versionsnummer sein, die allerdings bei jeder Änderung steigen muss. Es hat sich eingebürgert, hier das Datum in der Darstellung YYYYMMDD gefolgt von einer mehrstelligen Zahl zu verwenden, die hochgezählt wird, wenn am selben Tag mehrere Änderungen durchgeführt werden. In diesem Fall wurde die Datei zuletzt am 13.11.2001 und an diesem Tag das dritte Mal geändert.

Die anderen Zeilen geben folgende Informationen:

- [IN NS] Der zuständige Namensserver für die Domäne
- [IN MX Nummer] Der Mailserver. Es können mehrere Server angegeben werden. Die Nummer gibt die Rangfolge an. Je kleiner die Nummer, desto höher ist der Rang.
- [IN A] Definiert die IP-Nummer eines Hostnamens. Es wird nur einmal für jede IP-Nummer definiert. Weitere Namen werden per CNAME angegeben.
- [IN CNAME] Definiert für einen Rechner, der bereits mit IN A angegeben ist, einen weiteren Namen. Man spricht hier von einem Nickname, übersetzt Spitzname.

Es werden die Namen mail, gaston und powermac auf IP-Nummern abgebildet und für mail die Nicknames asterix festgelegt.

Bei den in dieser Datei vorkommenden Namen wird vom System der Domänenname ergänzt, wenn der Name nicht mit einem Punkt beendet wird. Darum steht hinter mail.willemer.edu ein Punkt. Taucht im Zusammenhang des Tests der Domänname doppelt auf (also so etwas wie max.willemer.edu.willemer.edu) dann fehlt sicher irgendwo ein Punkt.

Die Datei nr2name ist das Gegenstück zur Namenstabelle. Hier werden die Hostanteile der IP-Nummern auf Namen abgebildet. Bei einem Class B Netz würden also zwei durch einen Punkt getrennte Nummern in der linken Spalte stehen. Diese Umsetzung von Nummern auf Namen wird im Allgemeinen bei Berechtigungsprüfungen oder bei Anzeigen verbundener Rechner und ähnlichem benötigt.

@	IN SOA	mail.willemer.edu.	root.mail.willemer.edu.
		(
		1999062702	;serial
		360000	;refresh every 100 hours
		3600	;retry after 1 hour
		3600000	;expire after 1000 hours
		360000	;default ttl is 100 hours
)	
;		wer ist der zustaeendige Namensserver	
		IN NS mail.willemer.edu.	
137	IN PTR	mail.willemer.edu.	
144	IN PTR	gaston.willemer.edu.	

Testen

Das Testen einer DNS-Konfiguration erfolgt mit dem Befehl nslookup. Nach dem Aufruf meldet sich

nslookup mit dem zuständigen DNS-Router. Anschließend können Sie Hostnamen eingeben und nslookup zeigt die ermittelte IP-Nummer.

```
gaston > nslookup
Default Server:  www-proxy.KI1.srv.t-online.de
Address:  212.185.254.170

> www.willemer.de
Server:  www-proxy.KI1.srv.t-online.de
Address:  212.185.254.170

Non-authoritative answer:
Name:  www.willemer.de
Address:  212.227.118.90

> www.apple.de
Server:  www-proxy.KI1.srv.t-online.de
Address:  212.185.254.170

Non-authoritative answer:
Name:  www.germany.euro.apple.com
Address:  17.254.3.153
Aliases:  www.apple.de

> exit
```

Hier wurde in einer Internetsitzung nach den Adressen `www.willemer.de` und der Adresse `www.apple.de` gefragt. Beendet wird die Sitzung mit dem Befehl `exit`. Der wiederholt auftretende Kommentar »Non-authoritative answer« bedeutet, dass der angerufene Server (hier von T-Online) kein autorisierter DNS-Server der Domäne `willemer.de` oder `apple.de` ist.

nslookup kennt die Option `server`, mit der angegeben werden kann, welcher DNS-Server angefragt werden soll. Diese Möglichkeit kann genutzt werden, um festzustellen, ob der eigene DNS-Server seine Informationen auch an fremde DNS-Server exportiert.

Root-Cache Dateien

Für einen Cache Only Server brauchen Sie eine Ausgangsdatei, in der die Adressen der DNS-Server der Rootdomain. Diese Datei können Sie unter dem Namen `named.root` per ftp vom Server `rs.internic.net` mit der Adresse `198.41.0.7` aus dem Verzeichnis `domain` heruntergeladen. Diese Datei muss als Ausgangsdatei für den Cache verwendet werden. Hier ein Protokoll, wie man die Datei herunterlädt.

```
gaston # ftp rs.internic.net
Connected to rs.internic.net.
220-*****
220-*****
220-***** InterNIC Public FTP Server *****
220-*****
220-***** Login with username "anonymous" *****
220-***** You may change directories to the following: *****
220-*****
220-***** domain - Root Domain Zone Files *****
220-*****
```

```

220-***** Unauthorized access to this system may      *****
220-***** result in criminal prosecution.              *****
220-*****                                              *****
220-***** All sessions established with this server are *****
220-***** monitored and logged. Disconnect now if you do *****
220-***** not consent to having your actions monitored *****
220-***** and logged.                                  *****
220-*****                                              *****
220-*****
220-
220 FTP server ready.
Name (rs.internic.net:root): anonymous
331 Guest login ok, send your e-mail address as password.
Password:
230 User ftp logged in. Access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd domain
250 CWD command successful.
ftp> get named.root
local: named.root remote: named.root
500 'EPSV': command not understood.
227 Passive Mode (198,41,0,6,177,154)
150 Opening BINARY mode data connection for named.root.
100% |*****| 2769 3.91 KB/s 00:00 ETA
226 Transfer complete.
2769 bytes received in 00:00 (3.07 KB/s)
ftp> quit
221-You have transferred 2769 bytes in 1 files.
221-Total traffic for this session was 4586 bytes in 1 transfer.
221 Thank you for using the FTP service on rs.internic.net.
gaston #

```

Die Datei named.root enthält die DNS-Server der Rootdomain, also die letzte Instanz.

Änderungen zu BIND 8 und 9

Ab Version 8 verwendet BIND eine neue Konfigurationsdatei named.conf, die die bisherige Datei named.boot ersetzt. Inhaltlich bleiben die bisherigen Einträge erhalten, sie werden syntaktisch ein wenig aufpoliert und zu Gruppen zusammengefasst.

Zunächst wird in der globalen Gruppe options eingetragen, was den Server als Ganzes betrifft. Beispielsweise finden Sie hier den Verzeichnisnamen, in dem die weiteren Konfigurationsdateien stehen.

```

options {
    directory "/var/named";
    forwarders { 194.25.2.129; };
};

```

In diesem Abschnitt wird neben dem Verzeichnis auch noch der Namensserver genannt, an den Anfragen weitergeleitet werden sollen, die der hiesige DNS-Server nicht beantworten kann. Neu sind die geschweiften Klammern, in denen die Konfigurationen zusammen gefasst werden und das Semikolon am Ende jeder Definition.

Für jede Domäne, die der Server bearbeitet, werden zwei Zonen eingerichtet. Die eine dient der Namensauflösung, die andere der Nummernauflösung. Für die Domäne willemer.edu würden Sie

beispielsweise folgende Einträge definieren.

```
#Namenaufloesung
zone "willemer.edu" in {
    type master;
    file "willemer.edu";
};
```

```
#Nummeraufloesung
zone "109.168.192.in-addr.arpa" in {
    type master;
    file "named.192.168.109";
};
```

Hier wird definiert, dass die Namensauflösung für die Domäne willemer.edu in der Datei named.willemer.edu stattfindet. Dass diese Datei im Verzeichnis /var/named liegt, wurde bereits in den options festgelegt. Dasselbe geschieht mit der Nummernauflösung. Hier heißt die Datei named.192.168.109. In beiden Fällen ist der DNS-Server ein Master.

Der Cache wird in gleicher Weise eingetragen. Allerdings ist dieser nicht vom Typ master, sondern vom Typ hint:

```
# Einbinden der von rs.internic.net geholten root.named
zone "." in {
    type hint;
    file "named.root";
};
```

Die Änderungen in den eigentlichen Konfigurationsdateien sind bei BIND 8 nur gering. Neu ist, dass der TTL-Wert (Time To Live; engl. Lebenszeit) bereits am Anfang der Datei stehen muss. Dieser Wert spezifiziert, wann die Daten von einem anderem Server erneut gelesen werden müssen. In der Version 8 muss er noch nicht zwingend dort stehen, in der Version 9 wird der Eintrag allerdings verlangt. Er befindet sich entweder am Anfang der Datei vor dem SOA-Eintrag in der folgenden Form:

```
$TTL 1w
```

Oder Sie setzen den Wert im SOA Eintrag direkt vor das Schlüsselwort IN.

```
@      86400 IN SOA ns hostmaster (
```

Fehlt dieser Eintrag, finden Sie in den syslog-Protokollen (siehe S. syslog), also beispielsweise in der messages die Meldung »no TTL specified«.

Ein weiterer Fallstrick ist, dass die öffnende Klammer hinter dem SOA-Eintrag noch in der gleichen Zeile sein muss und nicht, wie oben der Übersichtlichkeit halber in der Folgezeile stehen kann.

Am TTL-Eintrag zu sehen ist auch die Möglichkeit, die Werte, die bisher nur in Sekunden definiert wurden, nun auch in Stunden (H), Tagen (D) oder Wochen (W) anzugeben.

Unterabschnitte

- [Aufbereiten der Dateien](#)
- [NIS-Server starten](#)
- [Starten eines NIS-Clients](#)

Network Information Service: NIS

NIS (Network Information Service) hat die Aufgabe, in einem UNIX Netzwerk die Informationen auf allen Maschinen konsistent zu halten. NIS verwaltet Konfigurationsdateien und stellt sie netzweit zur Verfügung. Dadurch ist es nur für Maschinen mit UNIX interessant, da andere Systeme beispielsweise mit der Datei /etc/passwd wenig anfangen können.

Die Konfigurationsdateien werden auf dem Zentralrechner, dem Master, in so genannte Maps umgewandelt. Anschließend kann der Client seine Informationen per Netz vom NIS holen. Da die NIS-Informationen auf dem Netz nicht besonders gesichert sind, eignet sich NIS nicht für Netze mit besonderen Sicherheitsanforderungen.

In den meisten Fällen wird man die Passwortdatei unter die Verwaltung durch NIS stellen, um die Benutzerkonten der Anwender über alle Maschinen konsistent zu halten. Die Datei /etc/passwd wird also zentralisiert. Damit haben Sie auf allen Rechnern im Netz das gleiche Passwort, und müssen es nur an einer Stelle verwalten.

Die Namen der Befehle von NIS beginnen meist mit yp. Das hängt damit zusammen, dass Sun NIS ursprünglich yellow pages, also Gelbe Seiten nennen wollte. Die britische Telefongesellschaft hatte allerdings ein Markenrecht auf dem Begriff, so dass Sun es auf NIS umbenennen musste.

Aufbereiten der Dateien

Im ersten Schritt muss festgelegt werden, welche Dateien per NIS verwaltet werden. Diese Dateien werden dann in die so genannten Maps überführt. Der zentrale Befehl dazu lautet makedbm, der allerdings normalerweise nicht direkt aufgerufen wird, sondern über ein Makefile, das sich im Verzeichnis /var/yp befindet. Hier werden später auch die Maps liegen. Ein Makefile ist eine editierbare Datei zum Generieren bestimmter Zieldateien und wird normalerweise in der Programmierung verwendet (siehe S. make).

Ändern Sie die Datei Makefile mit Ihrem Lieblingseditor. Sie finden hinter dem Label `all` : alle Dateien, die für das NIS umgestellt werden sollen. Mit dem Kommentarzeichen `#` können Sie die Dateien ausschließen, die Sie nicht mit NIS in dieser Domäne verwalten wollen.

```
# If you don't want some of these maps built, feel free to  
# comment them out from this list.
```

```
#all:  passwd group hosts rpc services netid protocols
```

```
# netgrp mail shadow publickey networks ethers
# bootparams printcap amd.home auto.master auto.home
# auto.local passwd.adjunct timezone locale netmasks
```

```
all: passwd group rpc services netid
```

Nachdem Makefile angepasst ist, werden durch einen Aufruf von make mit dem Parameter all die Map-Dateien erstellt. Auf manchen Systemen gibt es auch ein spezielles ypmake.

```
make all
```

Anschließend sollten im Verzeichnis /var/yp die Maps zu finden sein, mit deren Hilfe NIS die Anfragen später beantworten wird.

NIS-Server starten

Im nächsten Schritt wird die NIS-Datenbank für den Master initialisiert.

```
gaston# cd /var/yp
gaston# domainname willemer.edu
gaston# ypserv
gaston# /usr/lib/yp/ypinit -m
```

```
At this point, we have to construct a list of the hosts which
will run NIS servers.  gaston.willemer.edu is in the list of
NIS server hosts.  Please continue to add the names for the
other hosts, one per line.  When you are done with the
list, type a <control D>.
```

```
    next host to add:  gaston.willemer.edu
```

```
    next host to add:
```

```
The current list of NIS servers looks like this:
```

```
gaston.willemer.edu
```

```
Is this correct? [y/n: y]  y
We need some minutes to build the databases...
Building /var/yp/willemer.edu/ypservers...
Running /var/yp/Makefile...
gmake[1]: Entering directory `/var/yp/willemer.edu'
Updating passwd.byname...
Updating passwd.byuid...
Updating group.byname...
Updating group.bygid...
Updating rpc.byname...
Updating rpc.bynumber...
Updating services.byname...
Updating services.byservicename...
Updating netid.byname...
gmake[1]: Leaving directory `/var/yp/willemer.edu'
gaston#
```

Nach dem Start von ypinit fragt das Programm nach allen NIS Servern. Der eigene Rechner wird automatisch angezeigt. Hier können Sie weitere Slave-Server eintragen, die bei einem Ausfall des Masters einspringen. Wenn keine weiteren NIS Server in der Domäne arbeiten sollen, wird die Eingabe

mit der Tastenkombination ctrl-D abgeschlossen.

Damit kann dieser NIS Server bereits Anfragen beantworten. Passwörter haben eine besondere Position. Sie sollen leicht von den Benutzern auf jedem beliebigen Client geändert werden können. Änderung sollen dann sofort dem Masterserver aktualisiert werden. Das erledigt der zum NIS gehörige Passwortserver. Er wird wie folgt auf dem Masterserver gestartet:

```
rpc.yppasswdd -s /etc/yp/shadow -p /etc/yp/passwd -e chsh  
ypserv
```

Nun sollte ein Client mit dem Aufruf von yppasswd sein Passwort domänenweit ändern können.

NIS basiert auf dem RPC (Remote Procedure Call), der durch den Prozess portmap zur Verfügung gestellt wird. Dieser Prozess wird bei jedem auf RPC basierendem Server benötigt, da er die Umsetzung der Funktionsnummern durchführt. Das Programm rpcinfo hilft zu prüfen, ob der NIS-Server aktiv ist und sauber eingebunden ist.

```
gaston # rpcinfo -u localhost ypserv
```

```
program 100004 version 1 ready and waiting  
program 100004 version 2 ready and waiting
```

```
gaston #
```

Die oben angegebene Meldung zeigt an, dass der NIS-Server korrekt in das RPC eingebunden ist und läuft.

Starten eines NIS-Clients

Zunächst muss die Domäneninformation stimmen. Dies kann durch Aufruf des Befehls domainname erfolgen. Übrigens muss die Domäne des NIS nicht mit der des DNS übereinstimmen. Anschließend wird der Client-Dämon gestartet. Er heißt ypbind.

```
silver# domainname willemer.edu  
silver# ypbind -broadcast
```

Diese beiden Befehle sollten in den rc-Dateien des Systems stehen, damit das NIS nach dem Start bereits zur Verfügung steht. Wer Daten aus der /etc/passwd per NIS ermitteln will, sollte als letzten Eintrag in der lokalen passwd-Datei die folgende Zeile eintragen.

```
+:::~:::
```

Anschließend kann mit dem Befehl ypcat getestet werden, ob die Verteilung der passwd durch den NIS-Server funktioniert.

```
silver# ypbind -broadcast  
silver# ypcat passwd
```

Der Befehl ypcat muss eine Liste aus von Einträgen der passwd-Datei liefern, dann ist alles in Ordnung.

« [Domain Name Service: DNS](#) | [Namensauflösung](#) | [Netzgruppen: /etc/netgroup](#) »

Netzgruppen: /etc/netgroup

Netzgruppen sind eine besondere Zusammenfassung von Benutzern. Sie werden in der Datei /etc/netgroup definiert. Jede Netzgruppe hat einen Namen und wird durch ein oder mehrere eingeklammerte Tripel festgelegt, die aus Rechner, Benutzer und Domäne bestehen. Beispiel:

```
awfriends      (,arnold,) (gaston,,) (,,willemer.edu)
```

Dieser Eintrag bedeutet, dass awfriends aus allen Benutzern mit dem Namen arnold bestehen. Dazu kommen alle Benutzer eines Rechners gaston, ganz gleich in welcher Domain, und alle Rechner der Domäne willemer.edu.

Zu Netzgruppen werden Benutzer zusammengefasst, denen gemeinsame Rechte zugewiesen werden sollen. Beispielsweise können in den Konfigurationsdateien von NFS oder in den Dateien .rhosts der r-Kommandos auch Netzgruppen genannt werden. Um sie dort von Benutzern zu unterscheiden, wird ihnen das @-Zeichen vorangestellt.

Next Generation IPv6

Als die Internetnummern definiert wurden, hielt man die Anzahl der Adressen für extrem großzügig. Immerhin waren sie so dimensioniert, dass bei den damaligen Bevölkerungszahlen für jeden Menschen eine IP-Nummer verfügbar war. Erst wenn jeder Mensch einen eigenen Computer besäße, der ans Internet angeschlossen wäre, würden alle Adressen aufgebraucht sein. Diese Berechnung stellte sich bald als zu einfach heraus, da die Adressen netzweise vergeben wurden und dabei immer einige ungenutzte Nummern in Reserve gehalten wurden.

Als die Adressen mit der überraschend zunehmenden Verbreitung von Computern und Internetzugängen immer knapper wurden, begann man über eine Überarbeitung der IP-Nummern nachzudenken. So liest man heute überall, dass die nächste Generation von IP-Nummern vor der Tür steht. Nur wie weit sie von der Tür entfernt ist, scheint noch nicht ganz klar zu sein.

Eine Adresse unter IPv6 ist statt 4 Byte unter dem jetzigen IPv4 ein IPv5 gibt es nicht. dann 16 Byte oder 128 Bit lang. Es gäbe damit etwa $3,4 \cdot 10^{38}$ Adressen. Damit ergibt sich eine 24-stellige Zahl von Adressen pro Quadratmeter Erdoberfläche. Nemeth, Snyder, Seebass, Hein: UNIX-Systemverwaltung. Markt+Technik - Prentice Hall, München, 2001. S. 357.

Allerdings soll der Nummernumfang auch genutzt werden, um einige Dinge zu vereinfachen. Es wird immer noch eine Unterteilung in Netz- und Hostanteil der Adresse geben. Allerdings wird sie nicht wie bisher von einer Netzkategorie abhängig gemacht, sondern die Netzkennung wird in Zukunft konstant 45 Bit und Hostkennung wird immer 64 Bit lang werden.

Das Teilnetz wird nun in der IP-Adresse kodiert (Sub) und muss nicht mehr in den Routingtabellen verwaltet werden. Auch das führt zur Vereinfachung. Hinzu kommen zu Anfang 3 Bits zur Bezeichnung des Adresstyps (T). Die 64 Bit für die einzelnen Rechner sind so groß, dass man die MAC, also die Hardwareadresse der verwendeten Adapter darin ablegen kann. vgl. Nemeth, Snyder, Seebass, Hein: UNIX-Systemverwaltung. Markt+Technik - Prentice Hall, München, 2001. S. 363f

Die großen Anbieter haben die Umstellung als Prestigeobjekt betrachtet, und sie bereits recht weit vorangebracht. Ganz ohne Probleme wird eine solche Umstellung allerdings nicht ablaufen. Problematischer als die großen Systeme sind die einzelnen Programme und kleinere sowie ältere Systeme.

Die Umstellung zieht sich durch von Geräten wie Computer, Routern und Druckservern bis zu jedem einzelnen Netzprogramm wie ftp oder Webserver. Da die Bitzahl der IP-Nummern geändert wird, wirkt es sich auf alle Datenstrukturen aus, die IP-Nummern halten, bis hin zu Datenbanken, bei denen Tabellenstrukturen geändert werden müssen. Da viele Systeme nicht so schnell oder gar nicht umstellbar sind, gibt es einen Kompatibilitätsmodus, der vermutlich nach einer Umstellung noch recht lange aktiv sein müsste.

Aus diesem Grund ist es nicht ungewöhnlich, dass der Markt zurückhaltend auf IPv6 reagiert. Inzwischen haben Techniken wie das Masquerading (siehe S. masquerading) bewirkt, dass die

Katastrophenszenarien so schnell nicht eintreten werden. Es gibt sogar kompetente Aussagen, die besagen, dass eine Umstellung gar nicht erforderlich sei. Nemeth, Snyder, Seebass, Hein: UNIX-Systemverwaltung. Markt+Technik - Prentice Hall, München, 2001. S. 357f.

« [Netzgruppen: /etc/netgroup](#) | **Netzwerk** | [Grundausstattung Bordwerkzeug](#) »

Grundausrüstung Bordwerkzeug

UNIX stellt einige Werkzeuge für das Netzwerk zur Verfügung. Das Programm `ping` wurde bereits behandelt (siehe S. `ping`). Hier werden daneben noch weitere nützliche Helfer vorgestellt, die Sie kennen sollten.

Unterabschnitte

- [ICMP und ping](#)
 - [Verbindung zwischen Prozessen: netstat](#)
 - [Anzeigen der Netzwerkadapter](#)
 - [Anzeigen der Routingtabelle](#)
 - [Routen verfolgen: traceroute](#)
 - [HP-UX: lanadmin](#)
-

ICMP und ping

Die Verbindung zwischen zwei Rechnern wird am einfachsten über ping geprüft. ping gehört zur Protokollfamilie ICMP (Internet Control Message Protocol). ICMP-Pakete werden versandt, wenn

- das Gateway keine Kapazität zur Pufferung einer Nachricht hat
- ein Zwischenrechner erkennt, dass das Ziel unerreichbar ist
- ein Routingeintrag fehlerhaft ist, und das Senden über ein anderes Gateway sinnvoller ist.
- eine Prüfung der Strecke erforderlich ist.

Die Tatsache, dass ein Rechner auf ein Pingsignal reagiert, heißt nicht zwingend, dass er auch arbeitsfähig ist. So reagiert das OS/2 PC/TCP sogar noch, wenn der Rechner in den Systemabschluss gefahren wurde und noch nicht ausgeschaltet wurde. Aber es sagt genau das, was es soll, dass nämlich eine Netzwerkverbindung zu dem Rechner besteht und zumindest ein minimaler TCP/IP-Service läuft.

Mit ping können Sie recht schnell einige Problemen erkennen bzw. ausschließen. Hier sind ein paar Handgriffe und ihre Interpretation aufgezählt. Die Beispiele werden auf einem Computer mit der TCP/IP Nummer 192.168.109.144 namens gaston ausgeführt.

- [ping auf die eigene IP-Adresse]
Dieser Test ist gar nicht so unsinnig, wie er scheint. Es wird die Grundfunktion des lokalen TCP/IPs geprüft. Beispiel:

```
ping 192.168.109.144
```

- [ping auf IP-Adresse im eigenen Subnetz]
Ist der Anschluss an das Netz ok? Arbeitet die TCP/IP-Installation auf beiden Rechnern?

```
ping 192.168.109.99
```

- [ping auf den eigenen Hostnamen]
Je nachdem, ob der Name per /etc/hosts oder DNS aufgelöst wird, wissen Sie so, ob Sie eine funktionierende Namensauflösung besitzen.

```
ping gaston
```

- [ping auf einen Rechner hinter dem Router]
Meldet der eigene Rechner »no route to host«, stimmen die Routing-Tabellen nicht. Kommt die Meldung vom Router, hat er ein Problem. Kommt gar keine Antwort, wird der adressierte Rechner ein Problem mit dem Routing haben oder einfach abgeschaltet sein.

```
ping 10.4.4.4
```

- [ping mit großen Paketen]
Sie können ping durch die Option -s größere Pakete senden lassen. Die Größe wird als weiterer Parameter angegeben. Fallen dabei offensichtlich Pakete aus, gibt es Störsignale auf der Leitung oder die Fragmentierung eines beteiligten Rechners klappt nicht. Letzteres passierte auf alten SCO-Versionen, wenn ihnen die Netzwerkpuffer ausgingen. Typischer ist allerdings dieses Problem bei Koaxialkabeln, die oft Probleme durch Wackelkontakte insbesondere an den Abschlusswiderständen haben. Mit großen Paketen können Sie auch eine hohe Netzlast

simulieren.

```
ping -s 40000 192.168.109.143
```

Wenn ein Ping partout nicht durchkommt, kann es sein, dass eine Firewall, die zwischen Sender und Empfänger steht, die Pakete nicht durchlässt. Da ICMP-Pakete oft auch für Angriffe aus dem Internet verwendet werden, sind Firewalls inzwischen immer häufiger so konfiguriert, dass sie sie nicht mehr durchlassen. Dann funktioniert auch Ping nicht mehr. Anfällig gegen Angriffe mit ICMP-Paketen sind allerdings höchstens ältere Systeme.vgl. Ziegler, Robert: Linux Firewalls. Markt+Technik, München, 2000. S. 61.

« [Grundausstattung Bordwerkzeug](#) | [Grundausstattung Bordwerkzeug](#) | [Verbindung zwischen Prozessen: netstat](#) »

Verbindung zwischen Prozessen: netstat

Die Verbindung zwischen Prozessen wird mit Hilfe des Befehls `netstat` angezeigt. Bei jeder Verbindung werden die beiden Verbindungsendpunkte angezeigt. Ein Verbindungsendpunkt besteht aus einer IP-Adresse von dem verwendeten Port.

```
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 gaston.willemer.edu:ftp silver.willemer.ed:1072 ESTABLISHED
tcp        0      0 gaston.willemer.:telnet silver.willemer.ed:1071 ESTABLISHED
tcp        1      0 217.3.183.140:1253      195.30.193.73:www-http CLOSE_WAIT
tcp        1      0 217.3.183.140:1252      195.30.193.73:www-http CLOSE_WAIT
tcp        1      0 217.3.183.140:1251      195.30.193.73:www-http CLOSE_WAIT
```

Die Spalte `Local Address` zeigt die Sockets dieser Maschine und die Spalte `Foreign Address` die verbundenen Maschinen. Die Zahlen, die durch einen Doppelpunkt vom Host abgetrennt sind, sind die Sockets, die durch die Namen der Ports ersetzt sind, sofern sie in der `/etc/services` aufgeführt sind.

In der Spalte `State` steht der Status der Verbindung. Er ist `ESTABLISHED`, wenn die Verbindung angefordert wurde und von der anderen Seite bestätigt ist. `FIN` oder `FIN_WAIT` zeigt an, dass eine Verbindung abgebaut wurde. Beim Aufbau einer Verbindung sendet der Client eine `SYN`-Anfrage. Der Server sendet daraufhin ein `SYN-ACK`. Der Client sendet anschließend ein `ACK`-Data.

Beim Abbau von Verbindungen entsteht das Problem, dass der Port nicht sofort wieder vergeben werden kann. Ist also beispielsweise ein Server-Prozess abgestürzt, der eine Verbindung auf den well-known port hielt, kann er erst wieder gestartet werden, wenn die letzte Verbindung zu diesem Port aufgelöst wurde. Das bedeutet, dass alle Clients erst geschlossen werden müssen. Ist das Neustarten des Servers zeitkritisch, kann es sinnvoll sein, die Maschine kurz mit `ifconfig 1e0 down` oder gar durch Versetzen in den Single User Modus von den Verbindungsaufforderungen zu befreien.

Anzeigen der Netzwerkadapter

Mit dem Befehl `netstat -i` können Sie sich die vorhandenen Netzwerkschnittstellen anzeigen lassen. Dieser Befehl zeigt auch Informationen über die Pakete, die über die Schnittstelle gelaufen sind. Neben den statischen Ethernetadaptern gibt es aber auch virtuelle Adapter wie ISDN- oder Modemverbindungen, die nur im Augenblick einer Verbindung ansprechbar sind.

```
gaston> netstat -i
Kernel Schnittstellentabelle
Iface    MTU Met    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0     1500  0         0      0      0      0        4      0      0      0 BMRU
lo       16436  0       259      0      0      0       259      0      0      0 LRU
ppp0     1524  0        31      0      0      0        35      0      0      0 MOPRU

gaston> netstat -i
Kernel Schnittstellentabelle
Iface    MTU Met    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0     1500  0         0      0      0      0        4      0      0      0 BMRU
lo       16436  0       421      0      0      0       421      0      0      0 LRU
```

In der linken Spalte finden sich die Schnittstellen zum Netzwerk. In der nächsten Spalte steht die maximale Paketgröße des Adapters (MTU: Maximum Transfer Unit). Es folgen Statistiken über die versandten und die fehlerhaften Pakete. Diese Zahlen werden aufsummiert und sind bei Rechnern, die lange im Netz stehen, manchmal recht hoch.

Im Beispiel sehen Sie bei der ersten Anfrage ein `ppp0`-Device. Dies ist nur solange vorhanden, wie der Rechner Kontakt zum Internet hat. Beim zweiten Aufruf war die Verbindung bereits wieder geschlossen.

Das folgende Ausgabe stammt von einer HP-UX Maschine. Sie sehen hier, dass nicht aktuell aktive Devices zwar angezeigt werden aber nicht belegt sind.

```
hpsrv# netstat -i
Name Mtu Network Address Ipkts Ierrs Opkts Oerrs Coll
ni0* 0 none none 0 0 0 0 0
ni1* 0 none none 0 0 0 0 0
lo0 4608 loopback localhost 86 0 86 0 0
lan0 1497 192.168.109 hpsrv 75 0 63 0 0
hpsrv#
```

In den Spalten `Ipkts` und `Opkts` werden die Anzahl der Pakete seit dem Booten angezeigt. `Ierrs` und `Oerrs` sind dabei die Fehlerpakete. Dabei steht `I` für Input und `O` für Output. Damit kann der Datenverkehr genauer analysiert werden. Die Spalte `Coll` zeigt die Kollision von Paketen auf dem Netz. Bei Ethernet gehören Kollisionen zum Protokoll. Liegt der Anteil aber merkbar hoch, kommt es häufig zu Kollisionen. Dann sollten Sie versuchen, die Netzlast zu reduzieren.

Anzeigen der Routingtabelle

Zum Anzeigen und Kontrolle der eingetragenen Routen dient der Befehl `netstat` mit der Option `-r`.

```
gaston# netstat -r
Kernel IP routing table
Destination      Gateway         Genmask         Flags MSS window irtt Iface
192.168.109.0    *              255.255.255.0   U        0  0        0 eth0
loopback         *              255.0.0.0       U        0  0        0 lo
```

Dieser Rechner hat nur einen Netzadapter und loopback. Es gibt auch nur die eine Route auf sein eigenes Netz. Die angezeigten Flags haben die folgende Bedeutung

[Routingflags]C|L Flag & Bedeutung

U & used: die Route wird verwendet

G & gateway: die Route zeigt auf ein ganzes Netz über ein Gateway

H & host: diese Route zeigt nur auf einen einzelnen Rechner

D & dumped: wurde durch dynamisches Routen abgeschaltet

Routen verfolgen: traceroute

Der Befehl `traceroute` Der Befehl lautet `tracert` im PC/TCP für OS/2 und `tracert` unter MS Windows. liefert Informationen darüber, welche Gateways auf dem Weg von diesem Rechner zum Zielrechner passiert werden. Richtig interessant ist der Befehl natürlich, wenn man Internetadressen anfragt. Als Parameter nimmt der Aufruf Rechnername oder IP-Nummer. Als Beispiel wird die Route zur `www.willemer.de` verfolgt.

```
gaston# traceroute www.willemer.de
traceroute to www.willemer.de (212.227.118.90), 30 hops max, 40 byte packets
 1  217.5.127.105 (217.5.127.105)  59 ms  50 ms  50 ms
 2  217.5.127.94 (217.5.127.94)  50 ms  50 ms  50 ms
 3  FL-EB1.FL.DE.net.dtag.de (62.154.11.159)  50 ms  50 ms  50 ms
 4  F-gw12.F.net.DTAG.DE (62.154.17.194)  60 ms  60 ms  60 ms
 5  62.156.128.106 (62.156.128.106)  60 ms  60 ms  70 ms
 6  so-1100.gw-backbone-a.ka.schlund.net (212.227.112.85)  60 ms  60 ms  61 ms
 7  c1.gw-core-a.ka.schlund.net (195.20.224.19)  59 ms  70 ms  60 ms
 8  * kundenserver.de (212.227.118.90)  71 ms  70 ms
gaston#
```

Wenn bei der Routenverfolgung nur noch Sterne angezeigt werden, dann deutet das auf einen Fehler ab der letzten angezeigten, also funktionierenden Route hin. Interessant ist diese Information, um festzustellen, ob ein Problem innerhalb oder außerhalb der eigenen Routenverantwortlichkeit liegt und um eventuell weitere Schritte einleiten zu können.

HP-UX: lanadmin

HP-UX stellt mit dem Programm `lanadm` ein interaktives Programm zur Verfügung, mit dem Sie Statistiken auf der Paketebene sehen können. Nach dem Starten erreichen Sie ein Menü, von dessen Punkten `lan` interessant ist. Im nächsten Menü können Sie die Statistik wieder zurücksetzen. Mit dem Kommando `display` erhalten Sie eine Statistik. Mit `clear` werden die Statistiken zurückgesetzt.

TCP/IP-Dienste

TCP/IP bietet eine Vielfalt von Diensten. Der Umfang wird deutlich, wenn Sie einmal die Datei `/etc/services` betrachtet. In diesem Abschnitt werden die wichtigsten von ihnen vorgestellt.

Unterabschnitte

- [inet-Dämon](#)
- [File Transfer Protocol \(FTP\)](#)
 - [Der Client](#)
 - [Automatisieren des Einloggens](#)
 - [Konfiguration des FTP-Servers](#)
- [Anonymer FTP-Server](#)
- [TFTP, schnell und vertrauensvoll](#)
- [Terminaldienst \(telnet\)](#)
 - [telnet-Client](#)
 - [telnet-Dämon](#)
- [r-Kommandos](#)
 - [Die Datei .rhosts und hosts.equiv](#)
 - [Remote Copy \(rcp\)](#)
 - [rlogin](#)
 - [Befehlsausführung \(rsh, rcmd und rexec\)](#)
- [Wenn Sicherheit vorgeht: die ssh und scp](#)
 - [ssh als telnet-Ersatz](#)
 - [RSA-Authentifizierung](#)
 - [Tunnelbau: Andere Protokolle sichern](#)
- [NFS - Network File System](#)
 - [Konfiguration des NFS-Servers](#)
 - [NFS-Client](#)
 - [Festlegung in der /etc/fstab](#)
 - [Besonderheiten von MS Windows und OS/2](#)
- [Automatisches Mounten](#)
 - [Konfigurationsdateien](#)
 - [Prozesse und Hintergründe](#)
 - [Beispiel: Dynamisches Heimatverzeichnis](#)
 - [Perspektiven im LAN](#)
 - [Automount einer CD](#)

inet-Dämon

Die meisten Serverprozesse einer UNIX-Maschine werden relativ selten benötigt. Soll deren Dienst aber angeboten werden, muss ein Prozess ständig den Port auf Anfragen überwachen. Jeder dieser Prozesse benötigt Hauptspeicher. Zwar wird der Prozess bei Inaktivität bald in den Swapbereich verschoben, aber auch dieser ist nicht unerschöpflich. Zur Lösung dieses Problems gibt es den Internet-Dämon `inetd`, der mehrere Ports parallel abfragt. Sobald über einen dieser Ports eine Anfrage eintrifft, startet er den entsprechenden Server.

Gesteuert wird `inetd` durch die Datei `/etc/inetd.conf`. Hierin stehen die überwachten Dienste und die zugehörigen Serverprogramme. Jede Zeile hat den folgenden Aufbau:

Name Typ Protokoll Wartestatus UserID Server Argumente

Dabei bedeuten

- [Name] Name des Dienstes wie er in der Datei `/etc/services` steht.
- [Typ] Hier das Wort `stream` für jeden Dienst, der mit `tcp` und das Schlüsselwort `dgram` für Dienste, die mit `udp` in der Datei `/etc/services` stehen.
- [Wartestatus] Hier kann `wait` oder `nowait` stehen. Damit wird festgelegt, ob eine neue Anfrage des Dienstes wartet, bis die vorherige Anfrage ausgeführt ist. Steht hier `wait` können Anfragen nicht parallel bearbeitet werden.
- [UserID] Meist ist die UserID, unter der der Dämon läuft, `root`. Bei `finger` beispielsweise `nobody`.
- [Server] Hier steht der Dateiname inklusive Pfad des Serverprozesses.
- [Argumente] Anschließend folgt eine Kommandozeile des Serveraufrufs einschließlich Parameter.

Hier als Beispiel die Auszüge aus der `inetd.conf` für `ftp` und `telnet`:

<code>ftp</code>	<code>stream</code>	<code>tcp</code>	<code>nowait</code>	<code>root</code>	<code>/usr/sbin/tcpd</code>	<code>in.ftpd</code>
<code>telnet</code>	<code>stream</code>	<code>tcp</code>	<code>nowait</code>	<code>root</code>	<code>/usr/sbin/tcpd</code>	<code>in.telnetd</code>

Wenn Sie aus Sicherheitsgründen keinen Zugang für `telnet` oder `ftp` zulassen möchten, reicht es aus, wenn Sie den Eintrag in der `inetd.conf` mit einem Hashzeichen (`#`) auskommentieren. Änderungen in der Datei `inetd.conf` liest der Prozess `inetd` ein, wenn Sie ihm das Signal `SIGHUP` senden.

```
gaston# ps -ax | grep inetd
 566 ?        S      0:00 /usr/sbin/inetd
gaston# kill -SIGHUP 566
```


Unterabschnitte

- [Der Client](#)
- [Automatisieren des Einloggens](#)
- [Konfiguration des FTP-Servers](#)

File Transfer Protocol (FTP)

FTP dient zum Übertragen von Dateien auf einen fremden Rechner. Es ist aber nicht ein Verzeichnisdienst wie NFS oder im PC-Bereich Novell oder NT-Server, bei dem der Verzeichnisbaum des entfernten Rechners in den eigenen Baum eingebunden wird. Beim FTP werden einzelne Dateien explizit per Befehl übertragen.

Der Client

Für FTP gibt es einige Clients mit einer grafischen Oberfläche. Hier wird aber das Kommandozeilentool `ftp` mit seinen Kommandos betrachtet. Sie starten den FTP-Client durch Aufruf von `ftp`, gefolgt von der Internetnummer oder des Hostnamen, mit dem Sie in Verbindung treten wollen. `ftp` führt zunächst einen normalen Login aus. Sie werden also aufgefordert, Benutzernamen und Passwort einzugeben. Danach sind Sie auf dem fremden Rechner angemeldet und erhalten einen eigenen `ftp`-Prompt.

```
silver> ftp gaston
Connected to gaston.willemer.edu.
220 gaston.willemer.edu FTP server (Version 6.5/OpenBSD, linux port 0.3.2) ready.
Name (gaston:arnold): arnold
331 Password required for arnold.
Password:
230- Have a lot of fun...
230 User arnold logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> quit
221 Goodbye.
silver>
```

Vom `ftp`-Prompt können Sie wie aus einer Shell verschiedene Kommandos absetzen.

1. Allgemeine Befehle

- `[quit oder bye]`
Das Wichtigste zuerst: Mit `quit` verlassen Sie `ftp` wieder.
- `[cd]`
Sie können das Verzeichnis auf dem Zielrechner wechseln.
- `[lcd]`
Mit diesem Befehl wechseln Sie das Verzeichnis, in dem Sie sich lokal bewegen.
- `[ls oder dir]`
Anzeige der Dateien im aktuellen Verzeichnis auf dem Zielrechner
- `[pwd]`
Anzeigen, in welchem Verzeichnis Sie sich befinden.
- `[! Befehl]`

Damit rufen Sie eine Shell auf dem lokalen Rechner für einen Befehl.

Achtung: !cd entspricht nicht lcd. Da ! eine eigene Shell aufruft, wirkt der Aufruf von !cd nur auf jene Shell, die nach dem Aufruf wieder schließt und hat keine Auswirkung auf die ftp-Umgebung.

2. Dateien zwischen diesem und dem Zielrechner tauschen

o [get Datei]

Damit wird eine Datei vom fremden Rechner geholt. Der Dateiname muss exakt eingegeben werden. Wildcards werden nicht interpretiert.

o [put Datei]

Eine Datei wird auf den fremden Rechner geschoben. Auch hier muss der Dateiname exakt angegeben werden.

o [mget Dateien]

Das m bedeutet multi. Sie können * und ? verwenden, um mehrere Dateien anzugeben, die übertragen werden sollen. Nach jeder einzelnen Datei werden Sie gefragt, ob diese ebenfalls transferiert werden soll.

o [mput Dateien]

Hiermit werden mehrere Dateien auf den fremden Rechner geschoben, sonst wie mget.

Die Dateimaske für mput und mget kann Platzhalter (Wildcards) wie * und ? enthalten. Leicht verwendet man sie versehentlich bei get und put. Diese Befehle versuchen dann aber, eine Datei zu übertragen, deren Name die Platzhalter als Zeichen enthält.

3. Dateibearbeitung auf dem Zielrechner

o [delete Dateiname]

Lösche Datei

o [rename Original Ziel]

Benenne Datei um

o [mkdir Verzeichnisname]

Erzeuge ein Verzeichnis

o [rmdir Verzeichnisname]

Lösche ein Verzeichnis

4. Optionen

o [image oder ascii]

ftp arbeitet standardmäßig im ASCII-Modus. Das Programm stellt darin fest, welches Textformat auf der anderen Seite ist, und passt die Dateiendemarken während der Übertragung an. Während UNIX als Zeilenende das Zeichen Linefeed (ASCII Code 10) benutzt, verwendet ein Macintosh das Carriage Return (ASCII Code 13) und MS Windows eine Kombination aus beiden. Beim Übertragen einer Datei vom Mac auf UNIX würde in der Datei also jedes Byte, in dem 13 steht, durch 10 ersetzt. Das kostet nicht nur zusätzliche Zeit, das kann beispielsweise bei Datenbanken oder Programmen zerstörerische Folgen haben. Mit dem Befehl image oder dem Befehl binary schalten Sie diese Anpassung ab, mit ascii wieder an.

o [prompt]

Damit können Sie die Nachfrage bei den Befehlen mput und mget unterbinden. Wenn Sie mit Hilfe der Platzhalter * oder ? mehrere Dateien mit mput oder mget übertragen, fragt ftp nach jeder Datei, ob sie auch übertragen werden soll. Diese Abfrage schaltet der Befehl prompt ab. Ein weiterer Aufruf schaltet sie wieder an. Alternativ können Sie auch ftp mit der Option -i starten.

Daneben kennt ftp noch eine Reihe von Kommandos, die Sie natürlich in der Manpage finden. Letztlich reichen die vorgestellten Kommandos aus, um so ziemlich jeder Situation gewachsen zu sein.

Eine Sitzung, über die keine Daten mehr fließen, wird nach 900 Sekunden automatisch beendet.

Mit der Option -P kann ein anderer Port angesprochen werden, als der Standardport, wie er unter ftp in der Datei /etc/services steht. Das ist besonders wichtig bei dem Zugriff über Proxies (siehe S. proxy), da diese zum Weitertransport einer ftp Anfrage häufig einen anderen Port verwenden.

Automatisieren des Einloggens

In manchen Fällen soll der Austausch von Dateien automatisiert werden, also beispielsweise durch Programme oder Skripten gesteuert und auch zeitversetzt durch cron oder at (siehe S. cron) gestartet werden. In solchen Situationen stört die Passwornachfrage. Durch eine Datei namens .netrc im Heimatverzeichnis kann das Einloggen automatisiert werden. Dabei enthält diese Datei in jeder Zeile die Einträge:

Host Benutzerkennung Passwort

Zuerst wird der Host angegeben. Wird dieser Host später als Argument des `ftp` angegeben, werden Benutzerkennung und Passwort aus dieser Zeile verwendet. Der Eintrag des Passworts ist optional. Ist ein Passwort eingetragen, muss die Datei mit dem Kommando `chmod` auf 600 gestellt werden. Sie darf also von niemandem gelesen werden können, außer dem Besitzer.

Konfiguration des FTP-Servers

Der FTP-Server, den Sie standardmäßig auf jedem UNIX System vorfinden, heißt `ftpd` und wird normalerweise über `inetd` gestartet. Sie können ihn aber auch als Dämon aus den rc-Dateien starten. Dann muss `ftpd` allerdings mit der Option `-D` aufgerufen werden.

Da ein ungeschützter FTP-Zugriff durchaus sicherheitskritisch ist, kann der Zugang beschränkt werden. Wie auf Seite `stopftp` gezeigt, können Sie durch Auskommentieren des FTP-Dienstes in der Datei `inetd.conf` den FTP-Zugang komplett abschalten.

Existiert eine Datei namens `/etc/nologin`, wird diese einem Client angezeigt, wenn er sich anmelden will, und dann wird der Zugriff verweigert. Diese Methode ist ideal für kurzfristig abgeschaltete FTP-Zugänge oder wenn Sie beispielsweise darauf hinweisen wollen, dass der Zugang inzwischen auf einem anderen Rechner liegt.

Einzelne Benutzer können ausgeschlossen werden, indem Sie sie in der Datei `/etc/ftpusers` aufführen. Trotzdem der Dateiname andere Assoziationen erweckt, werden hier die Anwender aufgezählt, die keinen Zugriff auf den FTP-Server haben sollen.

« [inet-Dämon](#) | [TCP/IP-Dienste](#) | [Anonymer FTP-Server](#) »

Anonymer FTP-Server

Der normale FTP-Zugang erfolgt über das Anmelden als Benutzer des Systems. Im Internet gibt es viele FTP-Server, die öffentlich Dateien zur Verfügung stellen. Man spricht von anonymem ftp, weil man sich mit der Kennung »anonymous« anmeldet. Als Passwort pflegt man die eigene E-Mail Adresse anzugeben.

Ein Rechner kann nur entweder ein anonymes oder reguläres ftp anbieten. Ist ein anonymer Server aktiv, können Sie sich nicht mehr über das normale ftp einwählen. Ein solcher Zugriff wird abgewiesen mit dem Hinweis, dass nur anonymes ftp zugelassen ist.

Um ftpd als anonymen Dienst zu starten, wird der ftp-Dämon in der Datei inetd.conf bzw. in der rc-Datei mit der Option -A gestartet. Es muss außerdem ein Benutzer namens ftp auf dem System eingerichtet sein. Beim Login als »anonymous« werden die Benutzer in das Heimatverzeichnis der Benutzer ftp gesetzt. Dort wird intern ein chroot aufgerufen. Der Befehl bewirkt, dass für den Benutzer das aktuelle Verzeichnis zum Wurzelverzeichnis wird. Dadurch wird jeglicher Zugriff auf andere Bereiche des Verzeichnisbaums rigoros unterbunden.

```
# /etc/inetd.conf
ftp stream tcp nowait root /usr/sbin/tcpd in.ftpd -A
```

Es gibt noch andere FTP-Server, wie beispielsweise der WuFTP-Server, der vor allem in sicherheitskritischen Umgebungen gern eingesetzt wird. Beispielsweise ist es mit WuFTP möglich, bei normalen Benutzern per chroot den Zugriff auf andere Verzeichnisse als das eigene Heimatverzeichnis zu verhindern. WuFTP entstand an der Washington University in Saint Louis. Nähere Informationen finden Sie unter

<http://www.wu-ftpd.org>

TFTP, schnell und vertrauensvoll

Das Trivial File Transfer Protocol wird auch manchmal Trusted FTP genannt, weil es keine Authentifizierung verlangt. Der Zugriff ist auf Dateien beschränkt, deren Rechte auf dem System den Zugriff für alle Benutzer erlauben.

Auf den meisten Systemen ist TFTP in der `inetd.conf` abgeschaltet. Seine Hauptbedeutung erlangte TFTP im Zusammenhang mit Workstations, die keine eigene Festplatte besaßen, den »diskless workstations«. Diese haben per TFTP ihre Grunddaten von anderen Maschinen bezogen. Heutzutage findet man solche Geräte selten, und aus Sicherheitsaspekten ist TFTP wenig ratsam.

Der größte Unterschied zu FTP ist eigentlich in der Realisierung. TFTP basiert auf `udp`, während FTP ein `tcp` Dienst ist.

[« Anonymer FTP-Server](#) | [TCP/IP-Dienste](#) | [Terminaldienst \(telnet\)](#) »

Unterabschnitte

- [telnet-Client](#)
 - [telnet-Dämon](#)
-

Terminaldienst (telnet)

Mit dem Programm `telnet` können Sie eine Terminalsitzung über das Netz auf einem fernen Rechner aufbauen. Aus Sicht des Anwenders unterscheidet sich solch eine Sitzung kaum von einer Terminalsitzung.

Da `telnet` alle Tastendrücke und Bildschirmausgaben direkt über das Netz überträgt, werden auch Passwörter im Klartext über das Netzwerk übertragen. In sicherheitskritischen Umgebungen werden darum Zugriffe per `telnet` ausgeschlossen. Statt dessen wird `ssh` (secure shell) verwendet, das die gleiche Leistung bringt, nur dass die übertragenen Daten verschlüsselt werden (siehe S. `ssh`).

telnet-Client

Als Argument wird `telnet` die Internetnummer oder der Hostname des Zielrechners angegeben. Auf dem Zielrechner wird ein login wie bei einem normalen Terminal gestartet. Nach der erfolgreichen Anmeldung wird auf dem Zielrechner eine Shell eröffnet und der Benutzer kann so arbeiten, als hätte er eine Terminalsitzung auf der fremden Maschine eröffnet.

`telnet` leidet unter dem gleichen Problem wie alle Terminalemulationen. Leider sind das Verhalten der Originalterminals nicht immer so exakt von der Software nachgebildet. In vielen Fällen werden Terminals emuliert, die gar keine Funktionstasten hatten, als sie gebaut wurden.

Wie beim Terminal auch, wird die Terminalemulation über die Umgebungsvariable `TERM` und die Einträge in der `termcap` oder der `terminfo` gesteuert.

Mit `telnet` können Sie sehr gut textorientierte Protokolle testen, wie sie von den meisten Internet-Services verwandt werden (beispielsweise HTTP oder NNTP). Beim Aufruf wird hinter dem Hostnamen als weiterer Parameter die Portnummer des Services angegeben und Sie sehen die Meldungen des Servers im Klartext. Sie können antworten und so das Protokoll verfolgen. Als Beispiel sehen Sie auf der Seite `pop3telnet` eine Sitzung mit einem POP3-Server.

Eine `telnet`-Sitzung wird durch das Ausloggen auf dem Zielrechner beendet.

telnet-Dämon

In der Datei `/etc/securetty` wird hinterlegt, welche Terminals so sicher sind, dass sich auch der Anwender `root` dort einloggen darf. Im Normalfall werden die virtuellen Terminals, die `telnet` verwendet, dort nicht aufgeführt. Aus diesem Grund ist es meist nicht möglich, sich als `root` per

telnet direkt anzumelden. Statt dessen verwendet der Administrator eine »zivile« Kennung und wechselt mit dem Kommando su zum root.

Auch der telnet-Server wird über den inetd gestartet. Dementsprechend gibt es auch einen Eintrag in der inetd.conf. Soll es auf die Maschine keinen Zugriff per telnet mehr geben, wird der Eintrag einfach auskommentiert, indem ein # an den Anfang der Zeile gesetzt wird (siehe S. stopftp).

« TFTP, schnell und vertrauensvoll | TCP/IP-Dienste | r-Kommandos »

Unterabschnitte

- Die Datei `.rhosts` und `hosts.equiv`
 - Remote Copy (`rcp`)
 - `rlogin`
 - Befehlsausführung (`rsh`, `rcmd` und `rexec`)
-

r-Kommandos

Das ständige Authentifizieren beim Anmelden innerhalb eines LANs (Local Area Network; siehe S. 1an) ist in der Praxis sehr lästig. Darum gibt es die Möglichkeit, direkt auf die ferne Maschine zuzugreifen. Ein Anwender erteilt einem Benutzer einer anderen Maschine die Berechtigung, seinen Zugang zu benutzen, indem er eine Datei namens `.rhosts` anlegt. Darin werden die Computer und Benutzer aufzählt, die ein direktes Zugriffsrecht über die Kommandos `rcp`, `rlogin` und `rsh` haben sollen.

Nach einer korrekten Konfiguration arbeiten die Befehle, ohne das Passwort zu verlangen. Wie so oft stehen Bequemlichkeit und Sicherheit umgekehrt proportional zueinander. Darum sind diese Protokolle auch sinnvollerweise im LAN und nicht im Internet einzusetzen. Die drei r-Kommandos `rcp` (remote copy), `rsh` (remote shell) und `rlogin` (remote login) entstammen dem BSD und sind auf allen UNIX-Derivaten zu finden.

Die Datei `.rhosts` und `hosts.equiv`

Die r-Kommandos sollen Anwendern, die auf zwei Hosts ein Benutzerkonto haben, erlauben, sich selbst den direkten Zugang zu den eigenen Ressourcen auf dem anderen Rechner zu erlauben. Dazu legen Sie im Wurzelverzeichnis des eigenen Heimatverzeichnisses eine Datei namens `.rhosts` an. Sie darf nur für den Besitzer schreib- und lesbar sein. In dieser Datei stehen die Namen der Rechner, von denen ein Zugriff über `rlogin`, `rcp` oder `rsh` auf diesen Rechner erlaubt sein soll.

Da der Anwender vielleicht einen anderen Benutzernamen auf dem fremden Rechner haben könnte, ist es auch möglich, hinter dem Rechnernamen noch eine Anwenderkennung aufzuführen. Jeder eingetragene Benutzer hat die Berechtigung von dem fremden Rechner ohne Passwort auf dem lokalen Rechner einzuloggen und frei Daten hin und her zu kopieren.

Alternativ kann der Systemadministrator in der Datei `/etc/hosts.equiv` Maschinen aufführen, denen er eine pauschale Gleichstellung bzgl. `rlogin` und `rcp` einräumen möchte. Das bedeutet, dass alle Benutzer, die auf beiden Maschinen existieren, frei per `rlogin` und `rcp` arbeiten können. Beim Eintrag eines Rechners werden auch alle Drucker für diesen Rechner freigegeben. Diese Datei darf nur Schreib- und Leserecht für den Besitzer haben und muss `root` gehören. Eine solche pauschale Freischaltung wird aber normalerweise nur dann eingerichtet, wenn beispielsweise mit zwei parallelen Servern gearbeitet wird oder wenn man im privaten Netz ohne Internetzugang arbeitet. Beispiel für eine `/etc/hosts.equiv`:

+ pm7500
+@awfriends

Die erste Zeile erlaubt dem Anwender willemer auf gaston den freien Zugriff ohne Passwort. In der zweiten Zeile wird allen Anwendern von idex der freie Eintritt gestattet.

Die dritte Zeile enthält einen gravierenden Fehler: Hier steht ein Leerzeichen zwischen dem Plus und dem Hostnamen pm7500. Das einzelne + wird so interpretiert, dass jeder(!) Rechner freien Zugriff hat. Dass der alleinstehende pm7500 dieses Recht auch hat, versteht sich dann von selbst.

In der vierten Zeile wird der Netzgruppe awfriends (siehe S. netgroup) ein solcher Zugriff eingeräumt. Sie muss in der Datei /etc/netgroup definiert sein.

In allen Fällen müssen die Rechnernamen natürlich in der Datei /etc/hosts stehen oder durch einen Namensdienst aufgelöst werden.

Das Sicherheitsrisiko liegt nicht in erster Linie darin, dass neben dem Administrator auch Benutzer Rechte weitergeben können. Denn letztlich ist das Eintragen eines Rechnernamens in der .rhosts immer noch wesentlich besser als die Weitergabe des Passwortes. rlogin hat sogar gegenüber telnet den Vorteil, dass das Passwort bei der Kommunikation nicht über das Netzwerk übermittelt wird. Es kann also nicht abgehört werden. Das kleinere Problem besteht darin, dass wenn ein Zugang des Benutzers geknackt ist, die anderen Zugänge auch offen sind. Das hört sich schlimmer an als es ist. Normalerweise verwendet der Benutzer auf verschiedenen Rechnern des LAN vermutlich sowieso die gleichen Passwörter und wenn der Eindringling einen Zugang kennt, kennt er alle. Problematischer ist der Angriff mit einer gefälschten IP-Nummer. Ist in der .rhosts ein Notebook oder ein Rechner eingetragen, der nicht ständig im Netz steht, kann ein Angreifer seinerseits mit einem Notebook mit eben dieser IP-Nummer ins Netz gehen und sich direkt einloggen, ohne das Passwort zu kennen.

Remote Copy (rcp)

Mit dem Befehl rcp ist es möglich, über Rechnergrenzen hinweg zu kopieren. Dabei ist die Syntax nur wenig komplizierter als die des Befehls cp.

rcp Optionen Quelle Ziel

Lediglich die Argumentbeschreibung ist etwas verändert. Ist Quelle oder Ziel auf einem anderen Rechner, wird vor den Pfadnamen der Rechnername gefolgt von einem Doppelpunkt gesetzt. Dies kann bei der Quelle und auch beim Ziel erfolgen. Werden sowohl als Quelle als auch als Ziel fremde Rechner angegeben, ist rcp in der Lage, zu erkennen, dass es effizienter ist, die Kopie direkt zwischen den beiden Rechnern auszutauschen, als die Daten zu holen, um sie gleich anschließend wieder zu versenden.

```
rcp idex:/etc/hosts* ./test
```

Mit diesem Befehl werden vom Rechner idex alle Dateien, die auf die Maske /etc/hosts* passen, in

das im aktuellen Verzeichnis liegende test kopiert. test muss natürlich ein Verzeichnis sein, da anzunehmen ist, dass der Befehl mehrere Dateien kopieren wird. Werden die Dateien auf dem fremden Rechner nicht mit absolutem Pfad angegeben, wird das Heimatverzeichnis als Ausgangspunkt verwendet.

rcp verwendet als Serverdienst den rshd, also eine Remote Shell. Über diese wird auf der Gegenstation der rcp-Client mit der Option -f aufgerufen, mit dem die eigentliche Übertragung stattfindet. Die Option -f ist in der Manpage nicht dokumentiert, da sie ausschließlich dazu dient, den gegenüberliegenden rcp in eine Art »Servermodus« zu versetzen. Dem rcp auf der Gegenseite werden als Parameter gleich noch die Dateinamen mit gegeben. Es gibt also keinen rcpd.

rlogin

Mit dem Befehl `rlogin` eröffnen Sie eine Terminalsitzung auf dem Zielrechner unter Ihrer eigenen Benutzerkennung. Voraussetzung ist, dass Sie auf der fremden Maschine ebenfalls eine Benutzerkennung besitzen und die Datei `.rhosts` auf dem fremden Rechner den aktuellen Computer eingetragen hat. In diesem Fall wird kein Kennwort und kein Passwort ausgetauscht und damit kann es auch nicht abgehört werden. Versuchen Sie einen `rlogin` auf einen nicht entsprechend vorbereiteten Rechner, wird `rlogin` davon ausgehen, dass die Kennung die gleiche ist wie auf dem aktuellen Rechner und das Passwort für die Zielmaschine anfordern.

Sie können das Einloggen auf eine fremde Maschine noch weiter vereinfachen, indem Sie einen Link von `rlogin` erstellen und dem Link den Namen eines Rechners geben. `rlogin` prüft beim Aufruf, ob es unter einem anderen Namen aufgerufen wurde und verwendet dann diesen Namen als Zielmaschine.

```
silver> ln -s `which rlogin` gaston
silver> gaston
Last login: Thu Mar 21 09:52:09 from gaston.willemer.edu
Have a lot of fun...
gaston>
```

In der ersten Zeile wird ein symbolischer Link von dem Dateinamen gebildet, den der Befehl `which` liefert, mit dem Ziel `gaston`. Dieser zeigt dann auf den vollständigen Pfad von `rlogin`. Nun können Sie `gaston` aufrufen. Über den symbolischen Link wird `rlogin` gestartet. Das Programm `rlogin` stellt fest, dass der Name unter dem es gerufen wurde, `gaston` lautet und verbindet zu `gaston`.

Befehlsausführung (rsh, rcmd und rexec)

Das Programm `rsh` dient dazu, ein Kommando auf einem fremden Rechner ausführen zu können. Auf einigen Systemen hat `rsh` den Namen `rexec`. SCO nennt den Befehl `rcmd` (remote command). Der Name `rsh` ist hier für die restricted shell, einer lokalen Shell mit eingeschränkten Rechten vergeben.

rsh host Befehl

Gegenüber `rlogin` wird `rsh` eingesetzt, wenn Skripten oder einzelne Programme auf anderen Maschinen Prozesse gestartet werden sollen. Das kann beispielsweise im Zusammenhang mit einer

Fernsteuerung der Fall sein. Auch wenn ein Login aufgrund begrenzter Lizenzen ein weiteres Mal nicht möglich ist, kann man mit rsh oft noch ein paar Abläufe starten.

Mit der Option -l kann der Benutzer angegeben werden, unter dessen Kennung der Befehl ausgeführt werden soll.

« [Terminaldienst \(telnet\)](#) | [TCP/IP-Dienste](#) | [Wenn Sicherheit vorgeht: die](#) »

Unterabschnitte

- [ssh als telnet-Ersatz](#)
 - [RSA-Authentifizierung](#)
 - [Tunnelbau: Andere Protokolle sichern](#)
-

Wenn Sicherheit vorgeht: die ssh und scp

Es gibt boshafte Menschen, die Leitungen abhören. Dabei fällt ihnen bei `telnet`, `ftp` und anderen Protokollen das eine oder andere Passwort in die Hände. Und anstatt sich eine eigenen Linuxrechner zu kaufen, brechen sie in Ihre Systeme ein, nur um endlich einmal mit UNIX arbeiten zu dürfen. Damit diese Menschen zur Ehrlichkeit gezwungen werden, wurde mit SSH (Secure Shell) die Möglichkeit geschaffen, den Datenverkehr übers Netz zu verschlüsseln.

SSH hat als freie Software begonnen. Inzwischen hat der Autor Tatu Ylonen eine kommerzielle Version herausgegeben und pflegt die freie Version nicht mehr. Dies hat eine Gruppe von Programmierern übernommen, so dass es eine aktuelle, freie Version gibt, die OpenSSH heißt. SSH basiert auf Verschlüsselungsalgorithmen, die an dieser Stelle nicht behandelt werden. Der Fokus des weiteren Abschnitts wendet sich auf die Konfiguration einer solchen Umgebung.

Als Client gibt es `ssh`, das einem `rlogin` oder `telnet` entspricht, also die Möglichkeit bietet, eine Sitzung auf einer fernen Maschine abzuhalten. Als zweiten Client gibt es `scp`, der im Syntax vollständig dem `rcp` entspricht. Die Clients werden in der Datei `ssh_config` konfiguriert, die sich im Verzeichnis `/etc` oder in `/etc/ssh` befindet.

Der Server heißt `sshd`. Seine Konfigurationsdatei heißt `sshd_config` und befindet sich ebenfalls im Verzeichnis `/etc` oder `/etc/ssh`.

Es gibt verschiedene Arten, `ssh` zu betreiben. Sie können `ssh` als `telnet`-Ersatz verwenden. Dazu muss an der Standardkonfiguration nichts geändert werden. Der Vorteil liegt einfach darin, dass man den Datenverkehr nicht abhören kann, da er verschlüsselt ist. Wenn Sie verhindern möchten, dass man sich von einer beliebigen Maschine einloggen kann, können Sie diese Art des Zugriffs in der Konfigurationsdatei sperren und somit nur den Zugriff von explizit genannten Systemen oder Netzen erlauben.

Die zweite Variante arbeitet über die gleiche Methode wie die `r-Tools`, also die Dateien `.rhosts` oder `hosts.equiv`. Der Sicherheitsgewinn liegt darin, dass die Datenübertragung nun verschlüsselt ist. Wollen Sie, wie bei den `r-Tools` den Aufruf ohne die Bestätigung durch die Eingabe von Passwörtern realisieren, verwenden Sie eine Art von Fingerabdruck, den Sie zwischen den Maschinen austauschen. Zu guter Letzt gibt es noch eine Konfiguration, die eine Mischform aus der Authentifizierung per `rhosts`-Datei und Fingerabdruck darstellt.

Im Folgenden wird eine Konfiguration als `telnet`-Ersatz und dann per RSA-Authentifizierung vorgestellt werden.

ssh als telnet-Ersatz

Die einfachste Verwendung des ssh ist sein Einsatz als telnet-Ersatz. Dazu muss lediglich gewährleistet sein, dass in der Datei sshd_config der Wert der Variablen PasswordAuthentication nicht auf no steht. Wird dieses Schlüsselwort gar nicht erwähnt, ist das in Ordnung. Standardmäßig steht dieser Wert auf yes. Melden Sie sich von einer fremden Maschine über einen ssh-Client an, wird es wie beim telnet eine Aufforderung zum Einloggen mit Benutzernamen und Passwort geben. Allerdings wird von Anfang an verschlüsselt gearbeitet. Der Eintrag in der /etc/sshd_config lautet:

```
PasswordAuthentication yes
```

Das Abschalten der Option PasswordAuthentication ist möglich und sinnvoll, wenn beispielsweise ein Webserver, der im Zugriff des Internets steht, nicht von außen per ssh ansprechbar sein soll. Er kann so konfiguriert werden, dass er nur von internen Rechnern der Firma administriert wird.

Für MS Windows gibt es das freie Programm PUTTY, das einen ssh-Client und einen scp beinhaltet. Quelle:

<http://www.chiark.greenend.org.uk/~sgtatham/putty>

RSA-Authentifizierung

RSA ist ein asymmetrisches Kryptoverfahren, das der Schlüsselmanagement von SSH zugrunde liegt und ist nach den Entwicklern Rivest, Shamir und Adleman benannt. Rivest, Shamir, Adleman: A Method for Obtaining Digital Signatures and Public Key Cryptosystems. Communication of the ACM, Feb. 1978.

Soll von einer Maschine eine SSH-Verbindung aufgebaut werden können, muss sie vom Server eindeutig identifizierbar sein. Da eine IP-Nummer leicht zu ändern ist, reicht diese Art der Identifikation nicht aus. Man generiert mit dem Programm ssh-keygen zwei Schlüssel für eine Maschine. Der eine Schlüssel ist privat und der zweite ist öffentlich. Der öffentliche Schlüssel wird auf dem Server hinterlegt.

Im Beispiel soll der Rechner gaston per ssh oder scp auf den Rechner silver zugreifen können. Bisher sind beide Rechner per rsh und rcp erreichbar. Das soll aber nach der Installation von ssh aufgelöst werden. Mit dem Kommando ssh-keygen wird auf gaston ein Schlüssel erzeugt.

```
gaston> ssh-keygen
Generating public/private rsa1 key pair.
Enter file in which to save the key (/home/arnold/.ssh/identity)
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/arnold/.ssh/identity
Your public key has been saved in /home/arnold/.ssh/identity.pub
The key fingerprint is:
3b:a2:62:ed:02:ef:30:79:a1:4b:0b:b6:35:21:d8:f1 arnold@gaston
gaston>
```

Wie aus den Meldungen zu entnehmen ist, befinden sich die Dateien mit den Informationen im

Unterverzeichnis `.ssh` des Heimatverzeichnisses. Die Datei `identity.pub` enthält eine Zeile mit dem öffentlichen Schlüssel. In dieser Datei befindet sich nur die eine Zeile, die auf `arnold@gaston` endet. Diese Zeile wird auf dem Zielrechner an die Datei `.ssh/authorized_keys` angehängt.

```
silver> cd
silver> rcp gaston:.ssh/identity.pub .
silver> cat identity.pub » .ssh/authorized_keys
silver> rm identity.pub
```

Wenn die Konfigurationsdateien der `ssh`-Komponenten nicht verändert wurden, kann sich nun der Benutzer `arnold` von `gaston` aus auf dem Rechner `silver` anmelden, ohne ein Passwort einzugeben.

```
gaston> ssh silver
Last login: Mon Feb 25 00:04:06 2002 from mail.willemer.edu
Have a lot of fun...
silver>
```

Das Kopieren erfolgt ähnlich wie beim `rcp`, allerdings ist `scp` von Haus aus etwas geschwätziger. Das Programm zeigt einen Verlaufsbalken beim Kopieren an. Dieses unterhaltsame Feature können Sie allerdings auch mit der Option `-q` abschalten. Mit `-B` können Sie verhindern, dass `scp` plötzlich in einem im Hintergrund laufenden Prozess nach dem Kennwort fragt.

```
gaston> scp silver:/etc/passwd .
passwd      100% |*****| 2071      00:00
gaston>
```

Nach einer Standardinstallation von `ssh` finden Sie eine fast vollständig auskommentierte Datei `ssh_config`. Dabei sind die Defaultwerte hinter den Variablen angegeben. Ihr Inhalt sieht folgendermaßen aus:

```
Host *
# ForwardAgent no
# ForwardX11 no
# RhostsAuthentication no
# RhostsRSAAuthentication yes
# RSAAuthentication yes
# PasswordAuthentication yes
# FallBackToRsh no
# UserRsh no
# BatchMode no
# CheckHostIP yes
# StrictHostKeyChecking yes
# IdentityFile ~/.ssh/identity
# IdentityFile ~/.ssh/id_dsa
# IdentityFile ~/.ssh/id_rsa
# Port 22
Protocol 1,2
# Cipher blowfish
# EscapeChar
```

In der hier aufgestellten Umgebung würde man `RhostsRSAAuthentication` auf `no` stellen. Die Werte hinter `Host` bezeichnen die vom Client ansprechbaren SSH-Server. In diesem Fall sind das alle. Die Werte hinter `Protocol` bezeichnen, welche SSH Protokollversionen in welcher Reihenfolge unterstützt werden sollen.

Die Konfiguration für den Server `sshd` ist deutlich umfangreicher, so dass hier nur ein Ausschnitt aus der `sshd_config` betrachtet werden soll.

```
IgnoreRhosts yes
RhostsAuthentication no
RhostsRSAAuthentication no
RSAAuthentication yes
PasswordAuthentication yes
```

Mit diesen Werten ist das Einloggen ohne Passwort nur erlaubt, wenn eine Schlüsselübergabe stattgefunden hat. Alle `rhosts`-Varianten der Authentifizierung sind ausgeschlossen. Ein direktes Einloggen mit der Eingabe eines Passwortes ist aber erlaubt.

Tunnelbau: Andere Protokolle sichern

Mit `ssh` können Sie Netzverbindungen anderer TCP/IP-Dienste sichern. Die Basis ist eine gewöhnliche Sitzung mit `ssh`, der man allerdings die Ports zuordnet. Dazu gibt es die Option `-L`. Als weitere Parameter wird ein Tripel aus eigener Portnummer, dem Benutzer auf dem Zielrechner und dem Zielport angefügt. Als Beispiel wird eine Sitzung aufgesetzt, die vom Rechner `silver` zu `gaston` führt. Der Zielport ist 110, der für POP3 verwendet wird.

```
silver> ssh gaston -L 2002:gaston:110
Last login: Sun Jul 7 11:32:27 2002 from silver.willemer.edu
Have a lot of fun...
gaston>
```

Sobald diese Sitzung angemeldet ist, kann auf dem Rechner `silver` ein Mailclient aufgerufen werden. In der Konfiguration des Mailprogramms wird als Server `localhost`, also `silver` angegeben. Der Port wird von 110 auf 2002 umgestellt. Sobald Sie nun Ihre Mail abrufen, wird der Mailclient lokal auf `silver` den Port 2002 anfragen. Der ist aber durch die `ssh`-Sitzung über eine gesicherte Verbindung mit dem Port 110 von `gaston` verbunden. Dieser Tunnel bleibt so lange bestehen, wie die `ssh`-Sitzung besteht. Mit dem Ausloggen ist auch der Tunnel nicht mehr zugreifbar.

Da die Verbindung durch SSH getunnelt wird, kann sie überall da aufgebaut werden, wo eine solche Verbindung erlaubt ist. Damit hat die Freigabe des SSH zur Konsequenz, dass die Berechtigten auch beinahe jede andere Verbindung zwischen den Rechnern aufbauen können, für die SSH gestattet ist. Eine Firewall wie auch ein Proxy können die Verbindung zwar als SSH identifizieren. Es bleibt ihnen aber unsichtbar, welcher Protokolltyp darin getunnelt wird.

Unterabschnitte

- Konfiguration des NFS-Servers
 - NFS-Client
 - Festlegung in der /etc/fstab
 - Besonderheiten von MS Windows und OS/2
-

NFS - Network File System

Network File System heißt übersetzt Netzwerkdateisystem. Es handelt sich also um ein Dateisystem wie das einer Platte. Allerdings wird es nicht lokal auf einer Maschine eingebunden, sondern kann von allen Maschinen im Netz erreicht werden. Es ist also ein idealer Ort, um Daten abzulegen, die von mehreren Maschinen aus gebraucht werden.

Wie gewöhnliche Dateisysteme werden auch Netzdateisysteme mit dem Befehl `mount` in den Verzeichnisbaum integriert. Jede UNIX-Maschine ist in der Lage, sowohl als Client als auch Server aufzutreten. Um Zirkel zu vermeiden, in denen ein Rechner einem anderen Rechner Verzeichnisse zur Verfügung gestellt wird, die er von ihm selbst erhalten hat, können über NFS gemountete Verzeichnisse normalerweise nicht wieder exportiert werden. Das gilt nicht für Linux.

NFS basiert wie NIS (siehe S. nis) auf dem RPC (Remote Procedure Call) und verwendet das Protokoll UDP. Für RPC muss sowohl für den Client als auch den Server der `portmap` aktiviert werden, der unter UNIX das RPC implementiert. Server und Client müssen sich als Hostnamen ansprechen können, es müssen also Einträge für den jeweils anderen in der `/etc/hosts` stehen oder über einen Namensdienst wie DNS erreichbar sein.

Da der Eigentümer über die User-ID in den Dateien gespeichert wird, ist es wichtig, dass die Benutzer auf den verschiedenen Computern auch gleiche User-ID haben. Es empfiehlt sich auch, darauf zu achten, dass die Uhrzeiten auf den verschiedenen Rechnern nicht allzusehr auseinander laufen.

NFS wird nicht durch einen, sondern durch mehrere Serverprozesse getragen. vgl. Hunt, Craig: TCP/IP Network Administration. O'Reilly, Sebastopol, 1994. p. 204.

- [nfsd] Dies ist der Dämon, der die eigentlichen NFS-Anfragen der Clients beantwortet.
- [biode] Das ist der Block I/O Dämon. Er behandelt die Clientseite des NFS. Normalerweise laufen acht dieser Prozesse parallel.
- [rpc.lockd] Dieser Prozess behandelt die Sperranforderungen im NFS.
- [rpc.statd] Dieser Monitorprozess wird von dem Sperrdämon benötigt.
- [rpc.mountd] Der Mountdämon bearbeitet die Anfragen der Clients ein Verzeichnis einzubinden.

Diese Prozesse werden in den `rc`-Dateien beim Boot der Maschine gestartet.

Konfiguration des NFS-Servers

Die Konfiguration eines NFS-Servers erfolgt in der Datei `/etc/exports`. Dort geben Sie alle

Verzeichnisse an, die von anderen Hosts eingebunden werden dürfen. Diesen Pfadnamen können Einschränkungen zugefügt werden, die Rechner, Netze oder auch einzelne Benutzer betreffen. Auch Schreibrechte können auf diese Weise eingegrenzt werden. Der Inhalt einer exports könnte folgendermaßen aussehen:

```
/usr/src      *.willemer.edu
/home/arnold  @awfriends(rw)
/cdrom        (ro)
```

Die Zeilen bedeuten nacheinander:

- Das Verzeichnis /usr/src wird an alle Rechner der Domäne willemer.edu exportiert.
- Die Netzgruppe @awfriends (siehe S. netgroup) dürfen lesend und schreibend zugreifen.
- Das Verzeichnis /cdrom darf jeder lesen, aber niemand schreiben.

Nach Änderungen in der /etc/exports wird durch ein `kill -1` auf den `mountd` die Konfiguration neu geladen. Unter Linux wird das Neuladen der exports durch den Aufruf `kexportfs -r` erreicht.

NFS-Client

Der Client kann ein NFS-Laufwerk wie eine gewöhnliche Platte mit dem Befehl `mount` (siehe S. mount) in den Verzeichnispfad integrieren. Der Befehl dazu lautet:

`mount -t nfs Hostname:Pfadname Mountpoint`

- [Hostname] Der Name des Rechners, der das Verzeichnis anbietet.
- [Pfadname] Der Pfad auf dem NFS-Server
- [Mountpoint] Die Stelle, an der das Verzeichnis lokal eingehängt werden soll.

Beispiel:

```
mount -t nfs idfix:/home/ingres /mnt
```

Dadurch wird das Verzeichnis /home/ingres auf dem Host idfix auf dem eigenen Host über den Pfadnamen /mnt erreichbar. Ein Befehl wie `ls /mnt` zeigt den Inhalt des Verzeichnisses /home/ingres auf dem Host idfix.

Die Option zur Bestimmung des Dateisystemtyps ist von System zu System unterschiedlich:

[Optionen für das Mounten von NFS]L|L System & Option
Linux und FreeBSD & -t nfs
SCO & -f NFS
HP-UX & -F nfs

So würde der obige Aufruf unter SCO lauten:

```
mount -f NFS idefix:/home/ingres /mnt
```

Um zu ermitteln, welche Verzeichnisse ein NFS-Server anbietet, wird der Befehl `showmount` verwendet:

```
silver# showmount -e gaston
```

Festlegung in der `/etc/fstab`

Sie können NFS-Laufwerke automatisch beim Booten einbinden lassen. Sie erreichen dies durch einen Eintrag in der `/etc/fstab`. Als Dateisystemtyp wird `nfs` angegeben.

Im folgenden Beispiel soll der Benutzer `arnold` die Möglichkeit erhalten, vom Rechner `silver` aus auf sein Heimatverzeichnis auf dem Rechner `gaston` zuzugreifen. In seinem Heimatverzeichnis auf `silver` hat er sich für diesen Zweck ein Verzeichnis namens `gaston` angelegt. Die Datei `/etc/exports` auf `gaston` hat dazu folgenden Eintrag:

```
# /etc/exports
/home/arnold *.willemer.edu(rw)
```

Die Datei `fstab` auf `silver` hat folgenden Eintrag:

```
# /etc/fstab
gaston:/home/arnold /home/arnold/gaston nfs user 0 0
```

Statt der Gerätedatei für das Dateisystem steht der Hostname, gefolgt von einem Doppelpunkt und dem freigegebenen Verzeichnis. Dahinter steht der Einhängepunkt. Es folgt der Dateisystemtyp, wie er auch beim Befehl `mount` gegeben wird.

Der Eintrag `user` ist nur unter Linux möglich. Dadurch kann auch ein Anwender den Befehl `mount` ausführen.

Da dieser Eintrag in der `fstab`-Datei steht, braucht der Anwender nur die Quelle oder das Ziel anzugeben. Der jeweils andere Parameter und die Optionen werden automatisch hinzugefügt. Zum Beispiel:

```
silver> mount /home/arnold/gaston
```

Besonderheiten von MS Windows und OS/2

MS-DOS-Systeme und deren Nachfahren, wie MS Windows oder OS/2, sprechen Platten über Laufwerksbuchstaben an und besitzen keinen zusammenhängenden Verzeichnisbaum. Also werden auch NFS Verzeichnisse als Netzlaufwerke eingebunden. Das NFS verhält sich also hier aus Anwendersicht etwa wie ein Novell-Netzwerk.

OS/2 und MS Windows ab Version 95 sind zwar in der Lage, UNIX-Namen zu verarbeiten,

unterscheiden aber nicht zwischen Klein- und Großschreibung. Dagegen kann ein NFS-Laufwerk durchaus zwei Dateien in einem Verzeichnis halten, die sich nur durch Groß- und Kleinschreibung unterscheiden.

NFS-Server gibt es beispielsweise auch für OS/2. OS/2 ist aber nicht in der Lage, alle UNIX Eigenschaften von Dateien in seinem Dateisystem abzubilden. Darum sollten Sie solche Lösungen nur im Ausnahmefall verwenden.

« [Wenn Sicherheit vorgeht: die](#) | [TCP/IP-Dienste](#) | [Automatisches Mounten](#) »

Unterabschnitte

- Konfigurationsdateien
 - Prozesse und Hintergründe
 - Beispiel: Dynamisches Heimatverzeichnis
 - Perspektiven im LAN
 - Automount einer CD
-

Automatisches Mounten

Es ist nicht immer sinnvoll, alle NFS Dateisysteme ständig eingebunden zu halten, vor allem wenn sie nur selten benötigt werden. Es reicht oft aus, wenn das Dateisystem eingehängt wird, sobald auf die Daten zugegriffen werden soll. Genau diese Möglichkeit bietet `automount`. Zunächst muss definiert werden, unter welchem Verzeichnis welches Dateisystem eingehängt werden soll. Danach überwacht `automount` dieses Verzeichnis als Einhängpunkt für das Dateisystem. Das wird dadurch realisiert, dass `automount` einen virtuellen Dateisystemtreiber an dieser Stelle installiert. Beim Zugriff auf diesen Pfad wird im Hintergrund das vorher festgelegte Dateisystem in den Verzeichnisbaum eingehängt.

Damit eröffnen sich diverse Möglichkeiten. Zunächst müssen Dateisysteme des NFS nicht in die `fstab` eingetragen werden. Das hat wiederum den Vorteil, dass eine Maschine beim Booten nicht von einer anderen Maschine abhängig ist. In Kombination von NIS ist es möglich, das Heimatverzeichnis beim Einloggen in andere Rechner quasi mitzunehmen. Wie das funktioniert, wird später genau beschrieben. `automount` ist aber nicht auf Netze beschränkt. Auch Wechselmedien können so leichter gehandhabt werden. Man legt das Medium ein und beim ersten Zugriff wird es eingebunden. Das MacOS X verwendet `automount` in dieser Weise für seine Wechselmedien.

Konfigurationsdateien

`automount` verwendet mehrere Konfigurationsdateien. Die zentrale Steuerung erledigt die Masterdatei `/etc/auto_master` bzw. `auto.master`. Hier sind die Verzeichnisse aufgelistet, für die später je ein Automountdämon zuständig sein wird.

Für jeden der Einhängpunkte wird in der Masterdatei eine weitere Konfigurationsdatei genannt. Konventionsgemäß steht sie ebenfalls im Verzeichnis `/etc` und der Name beginnt mit `auto` und schließt mit dem Einhängpunkt. In ihr steht beschrieben, welche Unterverzeichnisse zur Verfügung stehen, die Optionen des Befehls `mount` und das Device oder die NFS-Quelle, also ähnlich wie in der Datei `fstab`.

Prozesse und Hintergründe

Der Automounter basiert auf einem virtuellen Dateisystem namens `AutoFS`. Daraus resultiert der Name des Startskripts `autofs` im Verzeichnis `/etc/init.d`. Unter Solaris dient er nur dem Start und Stop der Automounterdämons `automountd`. Unter Linux wird er auch mit der Option `reload` zum Neueinlesen der Konfigurationsdateien aufgerufen. Das Programm `automount` dient unter Solaris der Kommunikation mit dem Dämon, während es unter Linux der Dämon selbst ist.

Automount geht recht rigoros in der Umsetzung seiner Kontrolle seiner Einhängpunkte vor. In diesem Verzeichnis darf anschließend kein anderer Prozess mehr Verzeichnisse anlegen und vorher angelegte Verzeichnisse werden durch Automount unsichtbar.

Beispiel: Dynamisches Heimatverzeichnis

Um das oben erwähnte Heimatverzeichnis per Automount einzurichten, wird unter dem Verzeichnis /home ein zusätzliches Verzeichnis auto für den Automounter eingerichtet. Sie können natürlich gleich das Verzeichnis /home einbeziehen. Sie können dann aber in diesem Verzeichnis nicht mehr direkt und lokal ein Heimatverzeichnis anlegen.

```
# /etc/auto.master
/misc      /etc/auto.misc
/home/auto  /etc/auto.home
```

Nun wird in der Datei auto.home für den Benutzer andrea definiert, dass das eigentliche Heimatverzeichnis auf dem Rechner gaston liegt und von dort per NFS zu mounten ist.

```
# /etc/auto.home
andrea      -fstype=nfs          gaston:/home/andrea
```

Tatsächlich ist die Option nicht nötig, da der Automounter normalerweise davon ausgeht, dass es sich um ein NFS-Verzeichnis handelt.

Bei dieser Konfiguration befindet sich das Heimatverzeichnis von andrea in /home/auto/andrea. Ein wenig eleganter Weg ist es, dieses Verzeichnis bei der Benutzerverwaltung so einzutragen. Um dennoch den Pfad /home/andrea verwalten zu können, legen Sie einfach einen symbolischen Link.

```
ln -s /home/auto/andrea /home/andrea
```

Wenn der Benutzer andrea auf silver angelegt wird, muss man darauf achten, dass die gleiche UID wie auf gaston verwendet wird. Als Heimatverzeichnis wird /home/andrea verwendet. Loggt man sich von außen ein, wird sofort das Verzeichnis von gaston eingehängt und andrea hat die gleiche Umgebung wie auf gaston.

Perspektiven im LAN

Sie können die Konfigurationsdateien auch dem NIS (siehe S. nis) unterstellen und dann alle Heimatverzeichnisse von allen Rechnern im Netz über einen dedizierten Heimatverzeichnisserver versorgen. Da die Workstations der einzelnen Benutzer keine spezifischen Daten mehr verfügen, sind sie auch relativ leicht austauschbar. Ein zusätzlicher Vorteil ist die vereinfachte Datensicherung.

Automount einer CD

Mit Hilfe von automount können Sie sogar das lästige Einbinden von CDs in den Verzeichnisbaum umgehen. In der Masterdatei ist ein Eintrag für das Verzeichnis /misc. Die entsprechende Konfigurationsdatei hat folgenden Inhalt:

```
# /etc/auto.misc
cd      -fstype=iso9660,ro      :/dev/cdrom
#floppy  -fstype=auto, sync      :/dev/fd0
```

Sie können nun feststellen, dass durch den Aufruf von `ls` mit `/misc/cd` als Parameter das CD-Laufwerk durchstartet und der Inhalt der gerade erst eingelegten CD angezeigt wird.

Schwieriger ist da schon das Entnehmen der CD. Da die CD automatisch eingebunden wurde, sperrt der Entnahmemechanismus. Allerdings können Sie mit Hilfe des Befehls `eject` oder durch den Befehl `umount` die CD wieder freigeben. Alternativ können Sie warten, bis die automatische Freigabe durch den Automount erfolgt. Dies ist standardmäßig 5 Minuten. Sie können diese Zeitspanne verändern, indem Sie den Automountdämon mit der Option `-t` starten.

MacOS X verwendet Automount für alle Wechselmedien. Da ein Wechselmedium beim Mac durch Ziehen auf den Papierkorb ausgeworfen wird, gibt es so keine Umgewöhnung für den Anwender. Im Hintergrund wird einfach ein `eject` ausgelöst.

« [NFS - Network File](#) | [TCP/IP-Dienste](#) | [Allgemeines zum Internetanschluss](#) »

Allgemeines zum Internetanschluss

Natürlich können Sie mit UNIX ebenso ins Internet wie mit jedem gängigen anderen Betriebssystem auch. Wie groß der Aufwand wird, ist davon abhängig, in welcher Umgebung die Maschine steht. Es besteht ein grundsätzlicher Unterschied, ob Sie einen Heimarbeitsplatz per Modem an T-Online anschließen oder eine Workstation in einem Firmennetz.

Sollen Sie eine Workstation in einem Firmennetz einbinden, das bereits an das Internet angeschlossen ist, ist der Aufwand gering. Sie brauchen nur die IP-Nummer des Gateways als default-Route verwenden und den DNS-Server in der Datei `resolv.conf` eintragen.

Falls ein Proxy (siehe S. proxy) eingesetzt wird, müssen weder DNS noch die Routingtabelle konfiguriert werden. Sie müssen allerdings die entsprechenden Clientprogramme, vor allem den Webbrowser, anpassen. Für einen Proxy müssen Sie normalerweise die IP-Adresse oder den Namen des Proxyrechners und den Port angeben, über den der Proxy das jeweilige Protokoll entgegen nimmt.

Soll eine direkte Anbindung eines Linuxrechners an einen Provider erfolgen, hängt die Konfiguration des Anschlusses von Ihrer Umgebung ab. Zunächst muss festgestellt werden, ob Sie per Modem, ISDN oder DSL ins Internet wollen. Dabei verhält sich beispielsweise ein ISDN-Modem nicht wie eine ISDN-Karte, sondern eher wie ein Modem. Einige dieser ISDN-Modems sind übrigens per USB angeschlossen und haben proprietäre Protokolle. In solchen Fällen können Sie davon ausgehen, dass es auch nur einen Treiber für MS Windows gibt. Wollen Sie ein solches Gerät verwenden, sollten Sie genau prüfen, ob es auch unter Linux betrieben werden kann. Die Installation unterscheidet sich auch darin, ob Sie von der Konsole, von KDE oder GNOME den Zugang ins Internet freischalten wollen. Zu guter Letzt ist es auch von der Distribution abhängig, die Sie wählen. Alle Distributionen unterstützen Sie bei der Installation des Internetzugangs teils durch Installationsprogramme teils durch ausführliche Dokumentationen. Auch einige Internetprovider haben auf ihren Webseiten Informationen, wie Sie einen Linuxrechner konfigurieren müssen.

Kern jeder Installation ist das PPP (Point to Point Protocol). Dieses Protokoll kann über Modem, ISDN oder Ethernet gelegt werden und tauscht mit dem Provider Informationen aus. Der Anfrager sendet Benutzername und Passwort. Als Antwort kommt die zugeteilte IP-Nummer für das Internet. Die IP-Nummer ist nur für diese Verbindung gültig und kann bei der nächsten Verbindung bereits anders sein. Es wird nun ein Pseudodevice mit dieser IP-Nummer eingerichtet, das auf der eigentlichen Kommunikationshardware wie beispielsweise dem Modem basiert. Die default route wird auf dieses Pseudodevice umgelenkt und die Namensauflösung auf den DNS-Server des Providers gerichtet. Damit ist der Rechner im Internet.

Im Zuge der Anbindung laufen einige Skripte wie beispielsweise `ip-up` im Verzeichnis `/etc/ppp`. Diese Skripten sind interessant, weil Sie dort eigene Skripten starten können, die beim Einstieg ins Internet ausgeführt werden sollen. Mein Arbeitsplatzrechner holt beispielsweise bei jedem Verbindungsaufbau die E-Mail der gesamten Familie.

Für die Einrichtung benötigen Sie normalerweise folgende Informationen:

- Benutzererkennung und Passwort

- Telefonnummer des Anbieters
- Die TCP/IP-Adresse des DNS-Servers des Internet Service Providers

Mit Hilfe eines Masquerading-Pakets (siehe S. masquerading) kann der Linux-Rechner für das LAN zum Internetrouter werden. So können Sie mit mehreren Rechnern parallel surfen. Bei dem Paket isdn4linux und bei Modems mit dem Wähldämon diald ist es auch möglich, die Verbindung bei Bedarf zu öffnen. Da das Gateway nicht ermitteln kann, wann die Sitzung beendet ist, wird es so eingestellt, dass nach einer gewissen Zeit ohne Verkehr wieder aufgelegt wird. Bei ISDN kann das aufgrund der schnellen Verbindungserstellung recht häufig erfolgen, bei einem Modem lässt man sich lieber eine Weile Zeit. Aus zwei Gründen sollte man einen solchen Automatismus genau überwachen. Zunächst ist es bei einer fehlerhaften Konfiguration vor allem des DNS möglich, dass regelmäßig unnötige Verbindungen aufgebaut werden. Ferner gibt es immer mehr Programme, vor allem unter MS Windows, die aus den unterschiedlichsten Gründen Kontakt mit ihrem Hersteller aufnehmen. Manche Programme prüfen, ob Updates vorliegen, andere spionieren den Rechner aus und einige Viren aktualisieren sich inzwischen über das Internet.

« [Automatisches Mounten](#) | [Netzwerk](#) | [Dynamische TCP/IP-Nummern \(DHCP\)](#) »

Dynamische TCP/IP-Nummern (DHCP)

DHCP dient wie das ältere bootp dazu, die IP-Nummern eines Netzes an einem zentralen Ort zu verwalten. DHCP ist in RFC 2131 und 2132 beschrieben. DHCP findet seine Anwendung in großen Netzwerken, bei denen möglichst ohne Eingriffe des Administrators schnell mal neue Rechner eingehängt werden sollen. Insbesondere beim Einsatz von Laptops, die man schnell ins Netz einbinden will, ist dieses Verfahren sehr hilfreich. vgl. Nemeth/Snyder/Seebass/Hein: UNIX Systemverwaltung. Markt+Technik - Prentice Hall, München, 2001, S. 372-374.

Die DHCP-Implementierung des ISC (Internet Software Consortium) gilt als Standard. Zur Konfiguration verwendet der DHCP-Server des ISC die Konfigurationsdatei `/etc/dhcpd.conf`. Der Server `dhcpd` liest die Datei beim Starten und bricht den Start ab, wenn er Fehler entdeckt.

```
option domain-name "willemer.edu";
option domain-name-servers gaston.willemer.edu;
```

```
default-lease-time 600;
max-lease-time 7200;
```

```
authoritative;
```

```
log-facility local7;
```

```
ddns-update-style ad-hoc;
```

```
subnet 192.168.109.0 netmask 255.255.255.0 {
    range 192.168.109.10 192.168.109.20;
    option routers gaston.willemer.edu;
}
```

Mit `option domain-name` und `domain-name-servers` wird die Domänenumgebung festgelegt. Der Namensserver ist wichtig, da der neue Rechner eine passende Bindung zwischen Namen und IP erhalten muss. Die Ressourcen, die der DHCP-Server vergibt, sind nur geliehen. Darum spricht man hier vom Leasing. Der Parameter `default-lease-time` gibt die Zeit in Sekunden an, die die Ressourcen vergeben werden. Ist der vorgegebene Zeitrahmen überschritten, muss sich der Client um eine Verlängerung bemühen. So ist gewährleistet, dass nicht Ressourcen an Rechner vergeben sind, die sich längst nicht mehr im Netz befinden. Die gesetzte Option `authoritative` besagt, dass dieser Server der offizielle DHCP-Server des Netzes ist. Der `ddns-update-style` beschreibt die Form, wie die Abstimmung mit dem DNS erfolgt und muss zwingend in der `dhcpd.conf` definiert werden. `log-facility` betrifft den syslog-Dämon (siehe S. syslog). Damit wird festgelegt, welche Facility, also welche Quelle, die Fehler des DHCP-Servers aus Sicht des syslog haben. `local7` ist eine von den frei verwendbaren Quellenangaben.

Die `subnet` Definition beschreibt, dass für das Subnetz 192.168.109.0 mit der passenden Netzmaske der Bereich der IP-Nummern zwischen 10 und 20 verteilt werden kann. Der Router dieses Subnetzes ist `gaston`.

Nachdem die `dhcpd.conf` fertiggestellt wurde, sollten Sie den `dhcpd` von Hand starten, um zu sehen, ob und welche Fehlermeldungen es gibt.

```
gaston # dhcpd
Internet Software Consortium DHCP Server V3.0rc12
Copyright 1995-2001 Internet Software Consortium.
All rights reserved.
For info, please visit http://www.isc.org/products/DHCP
wrote 0 leases to leases file.
Listening on LPF/eth0/00:00:e8:59:88:0f/192.168.109.0/24
Sending on   LPF/eth0/00:00:e8:59:88:0f/192.168.109.0/24
Sending on   Socket/fallback/fallback-net
gaston #
```

Der Client braucht keine Information darüber, welcher Rechner im Netz der DHCP-Server ist. Sobald er bootet wird er per Broadcast im lokalen Netz nach dem passenden Server fragen.

Der DHCP-Server ist auch in der Lage, einem bootp-Client seine IP-Nummer anhand seiner Ethernetnummer zu vergeben. Nähere Informationen darüber, wie Sie den DHCP-Server konfigurieren, finden Sie in der Manpage von `dhcd.conf` und der mitgelieferten Dokumentation.

« [Allgemeines zum Internetanschluss](#) | [Netzwerk](#) | [E-Mail](#) »

E-Mail

E-Mail ist auf UNIX-Maschinen schon immer ein Standardkommunikationsmittel gewesen. Dabei war anfangs nur die Kommunikation der Anwender einer Maschine untereinander möglich. E-Mail ist ein fester Bestandteil eines UNIX-Systems, so dass sie auch für Systemmeldungen verwendet wird. So senden beispielsweise crontab und at (siehe S. cron) ihre Ausgaben nach wie vor per E-Mail an den Anwender. Auch Druckerprobleme werden dem Anwender per E-Mail zugesandt. Richtig interessant wird E-Mail aber erst rechnerübergreifend im Netzwerk.

Unterabschnitte

- [Format einer E-Mail](#)
 - [UNIX und Mail](#)
 - [SMTP \(Simple Mail Transport Protocol\)](#)
 - [Mailqueue](#)
 - [Verteilen der Post: sendmail -q](#)
 - [Weiterleiten der Post: aliases und forward](#)
 - [Lokale Mail lesen](#)
 - [POP3](#)
 - [IMAP](#)
 - [Post sammeln: fetchmail](#)
 - [Mailserver und Domain](#)
 - [Erstes Beispiel: Interne Firmenmail](#)
 - [Zweites Beispiel: Anbindung an das Internet](#)
 - [SMTP per sendmail](#)
 - [E-Mail einsammeln \(fetchmail\)](#)
 - [Automatisieren bei Internetverbindung](#)
-

Format einer E-Mail

E-Mails sind Texte, die ein bestimmtes Format haben, das in der RFC 822 beschrieben wird. Der erste Teil ist der Header, durch eine Leerzeile abgetrennt folgt der eigentliche Inhalt. Die erste Zeile des Headers beginnt mit From: gefolgt von einem Leerzeichen. Darauf folgt die E-Mail Adresse des Absenders. Die restlichen Zeilen beschreiben die Attribute der Mail. Der Feldname steht zuerst. Ein Doppelpunkt und ein Leerzeichen trennen den Inhalt ab, der bis zum Ende der Zeile reicht.

Typische Feldnamen sind:

[E-Mail Felder]L|L Feldname & Inhalt
 From: & E-Mail Adresse des Absenders
 To: & E-Mail Adresse des Empfängers
 Subject: & Das Thema der Nachricht
 Date: & Das Sendedatum
 Reply-To: & E-Mail Adresse, an die der Absender die Antwort wünscht (optional)

Bei der Übermittlung der Mail setzt jede Vermittlungsstation ihren Stempel vor den Header einer E-Mail. So sieht der Header einer E-Mail beim Empfänger beispielsweise so aus:

```
Return-Path: <stephan.mattescheck@galileo-press.de>
Received: from localhost (localhost [127.0.0.1])
    by gaston.willemer.edu (8.11.6/8.10.2/SuSE Linux 8.10.0-0.3) with ...
    for <arnold@localhost>; wed, 16 Jan 2002 13:35:42 +0100
Received: from pop3.web.de [217.72.192.134]
    by localhost with POP3 (fetchmail-5.9.0)
    for arnold@localhost (single-drop); wed, 16 Jan 2002 13:35:42 +0100 (CET)
Received: from [212.227.126.171] (helo=moutng1.schlund.de)
    by mx10.web.de with esmtp (Exim 4.02 #6)
    id 16QoC7-0005m8-00
    for arnold.willemer@web.de; wed, 16 Jan 2002 12:25:27 +0100
Received: from [212.227.126.150] (helo=mxng07.kundenserver.de)
    by moutng1.schlund.de with esmtp (Exim 3.22 #2)
    id 16QoC6-0004tv-00
    for arnold.willemer@web.de; wed, 16 Jan 2002 12:25:26 +0100
Received: from [212.79.176.2] (helo=gw-galileo-tops.tops.net)
    by mxng07.kundenserver.de with esmtp (Exim 3.22 #2)
    id 16QoC5-0002I8-00
    for arnold@willemer.de; wed, 16 Jan 2002 12:25:25 +0100
Received: from stephan (stephan.galileo-press.de [192.168.57.228])
    by gw-galileo-tops.tops.net (8.9.3/8.9.3) with ESMTP id MAA09037
    for <arnold@willemer.de>; wed, 16 Jan 2002 12:27:02 +0100
From: "Stephan Mattescheck" <stephan.mattescheck@galileo-press.de>
To: arnold@willemer.de
Date: wed, 16 Jan 2002 12:26:39 +0100
MIME-Version: 1.0
Subject: Katalog
Message-ID: <3C45717F.12078.6C8A48@localhost>
Priority: normal
X-mailer: Pegasus Mail for windows (v4.01)
Content-type: text/plain;
    charset=ISO-8859-1
Content-description: Mail message body
Content-Transfer-Encoding: 8bit
X-MIME-Autoconverted: from Quoted-printable to 8bit by gaston.willemer.edu ...
```

Status: R
X-Status: N

Aus dem Header können Sie Informationen über alle Zwischenstationen entnehmen, die die E-Mail durchlaufen hat. Selbst Details, wie dass der Absender das Programm Pegasus als E-Mail Client verwendet, können aus dem Header herausgelesen werden.

« [E-Mail](#) | [E-Mail](#) | [UNIX und Mail](#) »

UNIX und Mail

Eine UNIX-Maschine verfügt von Haus aus über ein lokales Mailingsystem. Das zentrale Programm ist traditionsgemäß `sendmail`, das mehrere Aufgaben übernimmt. Einerseits wartet `sendmail` als SMTP-Dämon im Hintergrund auf Aktivitäten am Port 25, und legt die empfangenen Mails in der Mailqueue ab. Es kümmert sich aber auch um die Verteilung der im Briefkasten liegenden Mails an die einzelnen lokalen Postfächer der Benutzer, und es sorgt dafür, dass Mail an andere Domänen an die entsprechenden Rechner weitergeleitet wird.

SMTP (Simple Mail Transport Protocol)

SMTP ist das wichtigste Protokoll, das verwendet wird, um Nachrichten zu versenden. Es wird in RFC 788 und RFC 821 beschrieben und standardmäßig unter der Portnummer 25 abgewickelt. Man könnte den Port 25 also als eine Art Briefschlitz der Maschine bezeichnen. Hier werden von außen Nachrichten eingeworfen, die an Benutzer dieser Maschine gehen. Die Benutzer der Maschine wiederum benutzen den lokalen Port 25, um Nachrichten an andere lokale Benutzer oder an Benutzer anderer Maschinen einzuwerfen. Alle Nachrichten werden in der Mailqueue zwischengelagert. Nachrichten an fremde Rechner reicht `sendmail` an die entsprechenden Rechner weiter.

Schickt allerdings ein fremder Rechner E-Mail, die für einen ganz anderen Rechner gedacht ist, gleicht `sendmail` den scheinbaren Fehler aus und leitet die E-Mail weiter. Leider können so auch Massenmails, die man auch Spam nennt, über den fremden Rechner verteilt werden. Hat der Spammer den Absender noch geschickt gefälscht, ist der missbrauchte Rechner der einzige, der zurück zu verfolgen ist. Dieses Verhalten des `sendmail`, das man »Relay« nennt, ist in den neueren Versionen von `sendmail` abgeschaltet.

Firmen, die Mailedienste im Internet anbieten, versuchen zu erreichen, dass nur ihre Kunden SMTP nutzen können. Da SMTP von Haus aus kein Passwort verlangt, gibt es zwei Ansätze, sicher zu stellen, dass der Auftraggeber berechtigt ist. Der eine Weg ist, dass nur diejenigen SMTP verwenden dürfen, die in einer bestimmten Zeitspanne vorher ihre Post per POP3 (siehe S. pop3) abgeholt haben. Da POP3 ein Passwort verlangt, ist der Nutzer identifizierbar. Der zweite Weg ist eine Erweiterung der SMTP um eine Passwortübermittlung.

Die Konfiguration des `sendmail` erfolgt in der Datei `/etc/sendmail.cf`. Es gibt kaum Literatur zu `sendmail`, die nicht auf die schwere Verständlichkeit dieser Datei eingeht. Aus diesem Grund gibt es diverse Ansätze, die Konfiguration zu vereinfachen. Ein Ansatz verwendet die Makrosprache `m4`. Die entsprechende Konfigurationsdatei ist dann `sendmail.m4`. Es ist allerdings meist gar nicht nötig, die `sendmail.cf` komplett zu verstehen, da man sie selten von Grund auf neu erstellt. Es reicht, die entscheidenden Einträge anzupassen. Die meisten Einträge sind durch die Grundinstallation korrekt gesetzt.

In den letzten Jahren sind einige Mail Transport Agents (MTA) entwickelt worden, die anstelle von `sendmail` installiert werden können. Die bekanntesten sind wohl `smail`, `qmail` und `postfix`. Als Argument für den Einsatz dieser Pakete statt `sendmail` wird im Allgemeinen die einfachere Konfiguration angeführt. Diese einfachere Konfiguration ist teilweise aber auch durch geringere Möglichkeiten erkaufte. Denn trotz der schwierigen Konfiguration ist die Verbreitung von `sendmail` ungebrochen, da `sendmail` extrem flexibel, sehr effizient und zuverlässig ist.

Mailqueue

Die Mailqueue ist ein Verzeichnis, in dem Mails vor ihrer Verteilung gesammelt werden. Alles was über den Port 25 (SMTP) eintrifft, wird hier erst einmal gelagert. Sie können die zum Versand anstehenden Mails mit dem Befehl `mailq` auflisten. Die Mailqueue ist ein festgelegtes Verzeichnis, normalerweise `/var/spool/mqueue`, in dem sich für jede Mail zwei Dateien befinden.

```
gaston> mailq
                               /var/spool/mqueue (1 request)
--Q-ID--  -Size-  ---Q-Time---  ----Sender/Recipient-
fb2AaPI01856      2511 Sun Dec  2 11:36 <arnold@willemer.de>
      8BITMIME
                               <willemer@t-online.de>
```

Zu dieser E-Mail gibt es zwei Dateien im Spoolverzeichnis der Mailqueue. Die eine Datei enthält die Versanddaten und stellt `qf` vor die Q-ID und die andere enthält die eigentliche Mail. Man erkennt sie daran, dass ihr Dateiname mit `df` beginnt. Im Beispiel sind das die Dateien `dffB2AaPI01856` und `qffB2AaPI01856`. Der Pfad der Mailqueue wird in der Datei `/etc/sendmail.cf` unter dem Schlüsselwort `QueueDirectory` festgelegt.

```
o QueueDirectory=/var/spool/mqueue
```

Um die obige Mail aus der Queue zu entfernen, müssen Sie als `root` folgenden Befehl absetzen: Als Administrator sollten Sie sich den Auftrag, eine Mail zu löschen, schriftlich geben lassen.

```
rm /var/spool/mqueue/?ffb2AaPI01856
```

« [SMTP \(Simple Mail Transport](#) | [E-Mail](#) | [Verteilen der Post: sendmail](#) »

Verteilen der Post: sendmail -q

Eine weitere wichtige Aufgabe des `sendmail` besteht in der lokalen Verteilung der Post. Der Aufruf `sendmail -q` bewirkt, dass die Mail der Mailqueue auf die Postfächer der lokalen Anwender verteilt wird oder an die Rechner versendet wird, deren Adresse rechts neben dem `@` steht.

Auch beim Versenden der lokalen Mail wird die Post zunächst in die Mailqueue gestellt, sodass eine Nachricht von `root` an `arnold` erst versendet wird, wenn `sendmail -q` das nächste Mal durchgeführt wird. Ohne weitere Angabe wird die Verteilung genau einmal durchgeführt. Die Mail soll aber meist in regelmäßigen Abständen verteilt werden. Anstatt den Aufruf von `sendmail` in die `crontab` zu stellen, können Sie einfach hinter der Option `-q` einen Zeitabstand angeben. Dann geht `sendmail` als Dämon in den Hintergrund und das Verteilen in diesen Abständen wiederholen.

Jeder Benutzer auf der Maschine hat ein Postfach. Dazu gibt es ein Verzeichnis (normalerweise `/var/mail` oder `/var/spool/mail`), in dem für jeden Benutzer genau eine Datei steht, deren Name der Benutzername ist. Neue Mails werden einfach hinten an diese Datei angehängt.

[« Mailqueue | E-Mail | Weiterleiten der Post: aliases »](#)

Weiterleiten der Post: aliases und forward

Beim Versenden von Nachrichten berücksichtigt `sendmail` die Aliasdatei `/etc/aliases`. Hier stehen Adressaten und ihre tatsächlichen Zielorte. Beispiel:

```
root:          arnold, root
aw: awilllemer@os2aw
gr: grossow@os2gr
programmers:   aw, meier@uni.gintoft.de, gr
owner-programmers: aw
```

Die erste Zeile sorgt dafür, dass E-Mails an `root` auch an `arnold` gesandt werden. Da ich normalerweise als `arnold` eingeloggt bin, erfahre ich nun sofort, wenn etwas im System schief läuft. Der Eintrag `root` sorgt dafür, dass die Mail aber weiterhin auch an `root` direkt gesandt wird. `aw` und `gr` sind Abkürzungen, die zu einfacheren Adressen führen.

Das Ziel `programmers` ist eine Mailingliste. Wird eine E-Mail an diese Adresse gesendet, wird diese E-Mail an alle hinter dem Doppelpunkt aufgeführten Personen versandt. Mailinglisten werden gern eingerichtet, wenn sich eine Gruppe zu einem bestimmten Thema austauschen möchte. Entsteht beim Senden an `programmers` ein Problem wird mit `owner-programmers` der Teilnehmer angegeben, an den die Fehlernachricht gesendet werden soll.

Werden Einträge in `/etc/aliases` geändert, wird dies mit dem Befehl `newaliases` oder `sendmail -bi` an den Mailservice weitergeleitet.

Jeder Benutzer kann seine lokale Mail an einen anderen Rechner weiterleiten, indem er im Benutzerverzeichnis eine Datei `.forward` anlegt und hier die Zieladresse z. B. `uwe@myserver` ablegt. Durch das Weiterleiten wird die E-Mail auf dem lokalen Rechner gelöscht.

Lokale Mail lesen

Das standardmäßig verfügbare Tool zum Lesen der lokalen Mail heißt einfach `mail`. Es überprüft das Postfach. Ist die Datei des Benutzers leer, beendet es sich sofort mit der Meldung, dass keine Post vorhanden ist.

```
gaston> mail
No mail for arnold
gaston>
```

Wenn Post da ist, erscheint eine Liste aller vorliegenden Mails. Jede Mail hat eine Nummer. Der Zeiger in Form eines Größerzeichens zeigt auf die erste neue E-Mail. Diese wird durch Eingabe der Return-Taste angezeigt. Die Anzeige der Mail erfolgt wie bei `more`.

```
gaston> mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/mail/andrea": 4 messages 1 new 4 unread
U  1 promotion5@amazon.de  Sun Feb  3 20:58 289/12731 "Noch mehr tolle Preis"
U  2 newsletter@jako-o.de  Thu Feb 14 14:54 110/3977  "Post von JAKO-O - Feb"
U  3 newsletter@jako-o.de  Thu Feb 14 15:25 102/3594  "Post von JAKO-O - Feb"
>N 4 promotion5@amazon.de  wed Feb 20 10:09 610/23184 "Alles versandkostenfr"
&
```

Mit den Tasten `+` und `-` können Sie durch die Mails vorwärts und rückwärts navigieren. Dabei wird jedesmal die nächste E-Mail angezeigt. Mit der Leertaste können Sie in der Nachricht weiterblättern. Mit einem `d` kann die aktuelle Mail gelöscht werden. `mail` wird mit dem Kommando `x` verlassen, wenn keine Änderungen übernommen werden sollen oder mit `q`, wenn beispielsweise gelesene Nachrichten als solche markiert und gelöschte Nachrichten tatsächlich aus dem Postfach entfernt werden sollen.

Mit `mail` können Sie auch Nachrichten versenden. Dazu rufen Sie `mail` mit der Adresse des Empfängers als Parameter auf. Es erscheint eine Zeile, in der Sie aufgefordert werden, den Subject, also den Betreff, anzugeben. Danach geben Sie Zeile für Zeile den Text ein und schließen mit `ctrl-D` oder mit einer Zeile, die nur einen Punkt enthält. Eine Korrekturmöglichkeit voriger Zeilen gibt es nicht.

Das Programm `mail` hat vor allem zwei Vorteile: Es ist auf jeder UNIX-Maschine verfügbar und es stellt keinerlei Ansprüche an das Terminal. Ansonsten bietet sich Ersatz mit `elm` oder `pine` für Terminalumgebungen oder Netscape und die diversen Mailclients, die mit den Desktops ausgeliefert werden, an. Alle haben die Eigenschaft, die Nachrichten aus der lokalen Mailbox in `/var/spool/mail` zu lesen und die ausgehenden Mails in der Mailqueue abzulegen.

Die Mailprogramme bieten zur Verwaltung der E-Mails oft eigene Ordner zur Sortierung der Post an. Diese werden normalerweise im gleichen Format wie das Postfach abgelegt, allerdings in einer eigenen Datei in einem eigenen Verzeichnis. So verwendet `elm` das Verzeichnis `~/elm` oder der Netscape Messenger das Verzeichnis `~/nsmail`.

POP3

POP3 ist das aktuell gängige Protokoll, mit dem ein E-Mail-Client seine Nachrichten beim Server abholt. Es ist in der RFC 1939 definiert. Im Prinzip arbeitet es fast wie ein ferngesteuertes lokales Mailprogramm. Es meldet sich über die in der /etc/passwd abgelegte Kennung an und liest das Postfach in /var/spool/mail aus.

Dieses Verfahren ist ideal für Telefonanbindungen, bei denen die Dauer der Verbindung berechnet wird. Der Client zieht seine E-Mail vom Server herunter, löscht sie dort und schließt die Verbindung. Das dauert normalerweise nur wenige Minuten. Die Bearbeitung der Mail erfolgt offline auf dem Arbeitsrechner des Anwenders.

Um einen POP3-Client zu konfigurieren, brauchen Sie den Namen oder die IP-Nummer des POP3-Servers. Manche Clients fordern noch die Angabe der Portnummer, die für POP3 standardmäßig 110 ist, wie man der /etc/services entnehmen kann. Ansonsten benötigt das Mailprogramm noch die Kennung und das Passwort auf der UNIX-Maschine. Der Client hat die Möglichkeit, nur die bisher ungelesenen Nachrichten abzuholen oder alle. Er kann ferner nach dem Download die Nachrichten stehen lassen oder löschen.

POP3 definiert nicht, wie Nachrichten gesendet werden. Das Senden wird üblicherweise über SMTP abgewickelt.

In einer reinen UNIX-Umgebung wird ein POP3-Server gar nicht gebraucht, da die E-Mails von den Systemen per SMTP durchgeschoben werden. Darum gehört POP3 nicht zur Standardinstallation einer UNIX Maschine. Soll aber die Post durch andere Arbeitsplatzrechner gelesen werden, die selbst keinen SMTP Server anbieten, wie MS Windows oder Mac, dann installiert man auf dem Mailserver einen POP3-Server. Die Installation läuft normalerweise ohne weitere Konfiguration ab, da der POP3-Server die Authentifizierung ganz normal über die passwd-Datei (siehe S. passwd) laufen lassen kann. Die E-Mails können direkt dort abgeholt werden, wo sie der lokale Mailedienst ablegt.

Wie die meisten Internetprotokolle basiert auch das POP3 auf dem Senden und Empfangen einfacher Zeilen. Sie können solche Protokolle mit Hilfe des Programms `telnet` leicht anschaulich machen oder prüfen. Im Folgenden wurde eine kleine Sitzung protokolliert. Sie besteht aus Anmeldung, Anzeige, wieviele E-Mails vorliegen, herunterladen einer E-Mail und Ende der Sitzung. Die eigenen Eingaben sind fett gedruckt.

```
gaston> telnet gaston 110
Trying 192.168.109.144...
Connected to gaston.
Escape character is '^]'.
+OK ready <3745.1014213784@gaston.willemer.edu>
User andrea
+OK Password required for andrea.
pass daswerdeichhierauchgeradeimklartextschreiben
+OK andrea has 4 visible messages (0 hidden) in 44471 octets.
list
+OK 4 visible messages (44471 octets)
1 12980
2 4064
3 3673
```

```
4 23754
.
retr 1
+OK 12980 octets
Return-Path: <ems+HA564Q8DYULUE6@bounces.amazon.com>
Received: from localhost (localhost [127.0.0.1])
...
Mit vorzüglicher Selbstbeherrschung
Roswita Presswurst
```

```
.
quit
+OK Pop server at gaston.willemer.edu signing off.
Connection closed by foreign host.
gaston>
```

Die fett gedruckten Zeilen wurden als Kommandos in `telnet` direkt eingegeben. Im normalen Betrieb übermittelt sie der POP3-Client. Das erste Kommando `user andrea` meldet den Benutzer an. Der Benutzername wird vom Server aus der `/etc/passwd` entnommen. Als nächstes wird das Passwort gesendet. Der Befehl `pass` leitet es ein. Der Server bestätigt mit `OK` die korrekte Anmeldung und gibt an, dass 4 Nachrichten vorliegen, die zusammen 44.471 Bytes Oktett ist der von humanistisch gebildeten Informatikern präferierte Ausdruck für Byte. Das Wort leitet sich vom dänischen Wort `otte` für acht her. belegen. Mit dem Befehl `list` erhält der Client eine Liste von 4 Nachrichten mit deren Größen. Der einzelne Punkt schließt die Liste ab. Der Befehl `retr 1` holt die erste Nachricht. Sie erscheint im Klartext, beginnt mit dem Header und endet mit einem einzelnen Punkt. Zuletzt wird die Sitzung durch das Kommando `quit` beendet.

« [Lokale Mail lesen](#) | [E-Mail](#) | [IMAP](#) »

IMAP

IMAP4 ist definiert in RFC 1730, IMAP4r1 in RFC 2060. IMAP (Internet Message Access Protocol) ist eine Software zur Verwaltung von E-Mail, die an einer Schwachstelle von POP3 ansetzt. Wenn der Anwender mehrere Computer hat, dann verteilen sich auch seine Nachrichten auf diese. Dabei landen sie immer auf dem Rechner, der die Nachrichten heruntergeladen hat. In einer Firma wäre es wünschenswert, wenn man seine E-Mail von jedem beliebigen Arbeitsplatz aus bearbeiten könnte. Auch wer oft unterwegs ist, hätte gern seine E-Mail sowohl auf dem Laptop als auch im Büro vollständig und aktuell.

Diese Situation kann man lösen, indem man die E-Mail auf dem Server lässt. Hier setzt das Protokoll IMAP an. Nicht nur die Daten liegen zentral, sogar die Ordner, die sich die meisten Anwender auf ihrem Client anlegen, um ihre Post zu sortieren, werden auf dem Server gehalten. Unter IMAP heißen diese Ordner Mailboxen. Da der Client die Daten niemals herunterladen muss, können Sie von jedem Arbeitsplatz mit einem IMAP-Mailclient die E-Mail bearbeiten. Sie können den Gedanken auch weiter treiben. Eigentlich brauchen Sie nicht einmal zwingend einen Mailclient. Es reicht, wenn Sie eine Intranetanwendung haben, die Sie mit dem Webbrowser aufrufen können.

Für Firmen ergibt sich ein weiterer Vorteil bei der Frage der Datensicherung der Mails. Das ist natürlich wesentlich einfacher, wenn die Daten zentral abgelegt werden, als wenn man die Postfächer auf den verschiedenen Arbeitsplätzen sichern muss.

Es gibt zwei relevante Implementationen eines IMAP-Servers. Die erste stammt vom IMAP-Entwickler Mark Crispin von der Universität Washington und wird als UW-Server bezeichnet. Die andere Implementierung ist von der Carnegie Mellon University und läuft unter dem Namen »Projekt Cyrus«. Der UW-Server braucht nach einer Installation auf einem UNIX-Rechner keine weitere Konfiguration, wenn sendmail als MTA (Mail Transport Agent) verwendet wird. Dianna und Kevin Mullet schreiben, dass sich der UW-Server nicht mit dem Maildir-Format von Qmail verträgt. vgl Mullet/Mullet: Mailmanagement mit IMAP. O'Reilly, 2001. S. 255. Der Cyrus-Server bietet die Möglichkeit, Mailquotas zu setzen. Diese Fähigkeit ist vor allem wichtig für Internetprovider, die einem Benutzer nicht erlauben können und wollen, dass ihre Mailbox beliebig groß wird. Dieser Vorteil wird durch einen größeren Installations- und Wartungsaufwand erkauft. So müssen die Mailboxen auf das Cyrus-System umgestellt werden und eine eigene Benutzerverwaltung aufgesetzt werden. Für Provider hat letztere den Vorteil, dass die /etc/passwd nicht mit reinen Mailbenutzern aufgebläht wird und dezentral verwaltet werden kann.

Inzwischen sind die meisten neueren Mailprogramme auch mit einer Unterstützung für IMAP ausgestattet. An plattformübergreifender Mailsoftware bietet sich der Netscape Messenger und das Mailprogramm von StarOffice an. Unter UNIX sind die terminalorientierten Programme mutt und pine mit IMAP ausgestattet. Unter X sind die neueren Mailprogramme, die beispielsweise mit den Desktops geliefert werden, wie etwa KMail (KDE) und Evolution (GNOME) verwendbar. Unter MS Windows funktionieren fast alle Mailprogramme wie Pegasus, Outlook und Outlook Express. Für Mac ist ebenfalls der Netscape Messenger verfügbar, aber auch das weit verbreitete Eudora kommt mit IMAP zurecht. Dies sind nur Beispiele. Sie werden feststellen, dass fast jede aktuell entwickelte Mailingsoftware neben POP3 auch IMAP beherrscht.

Für den Privatkunden, der eine Wahlleitung zum Internet hat, ist der Online-Modus von IMAP nicht sinnvoll. Der Mailclient verfügt bei der Trennung der Leitung nämlich nur über die Header der E-Mails.

Für das Lesen jeder E-Mail muss erneut eine Verbindung zum Server hergestellt werden. Da dies bei einer Wählleitung praktisch kaum möglich ist, gibt es auch noch den Offline Modus, bei dem die Mails auf dem lokalen System gelagert werden.

Zu vollständigen Speicherung des Mailverkehrs eines Anwenders benötigen Sie auf dem Server beträchtliche Kapazität. Das ist im Firmennetz kein Problem, da es fast egal ist, ob diese Daten auf dem lokalen Anwenderrechner liegen oder zentral abgelegt werden. Für einen Provider ist es schon schwieriger, diese Datenmengen für jeden Kunden bereit zu stellen. Wenn einige Kunden dann auch noch mp3-Dateien oder Videodateien senden und empfangen, wird der Service schnell unbezahlbar. Einer der großen Freemailldienste im Web bietet einen IMAP-Dienst an. Er ist aber auf 8 MByte beschränkt. Der Umfang meiner E-Mail-Daten liegt inzwischen bei dem Zehnfachen und es sind keine der genannten Speicherfresser dabei.

Zusammenfassend kann man sagen, dass IMAP gegenüber POP3 eine bessere Lösung für das Intranet (siehe Glossar S. intranet) ist. Solange die meisten Internetnutzer keine FlatrateFlatrate bedeutet, dass die Verbindungsgebühren pauschal monatlich bezahlt werden und nicht minutenweise abgerechnet wird. haben und solange der Speicher im Internet nicht großzügiger bemessen ist, wird POP3 hier sicher nicht so schnell abgelöst werden.

« [POP3](#) | [E-Mail](#) | [Post sammeln: fetchmail](#) »

Post sammeln: fetchmail

Große Firmen betreiben einen eigenen Mailserver, der direkt im Internet steht. Dieser bekommt die eingehende E-Mail per SMTP zugesandt. Dazu brauchen Sie eine Standleitung zum Internet und eine feste Internetadresse. Für kleinere Firmen lohnt sich dieser Aufwand oft nicht. Immerhin können Sie mit einem POP3-Postfach beinahe die gleiche Wirkung erreichen.

Die Lücke schließt das Programm `fetchmail`. Es holt bei Aufruf aus einer Liste von POP3-Postfächern die E-Mail ab und stellt sie in das lokale Mailsystem per SMTP ein. Dabei ist frei einstellbar, welches Postfach zu welchem lokalen Anwender gehört. Der Aufruf kann durch die `crontab` (siehe S. cron) in regelmäßigen Abständen angestoßen werden oder beispielsweise jedesmal, wenn sowieso eine Verbindung zum Internet geöffnet wird.

Konfiguriert wird das Programm über die Datei `.fetchmailrc` des jeweiligen Benutzers. Sie muss im jeweiligen Heimatverzeichnis des Aufrufers stehen. Eine Zeile in dieser Datei hat folgenden Aufbau:

poll Server protocol POP3 user User password passwd is localUser

Leider muss hier das Passwort im Klartext stehen. Damit nicht jeder die Passwörter auslesen kann, muss die Datei mit `chmod 600 .fetchmailrc` für Fremde unlesbar gemacht werden.

Der folgende Aufruf fragt alle aufgeführten POP3-Server ab und die E-Mails werden ins eigene Mailsystem gestellt.

```
fetchmail -k -L /fetchmail.log
```

Damit die E-Mails auch in den lokalen Briefkästen landen, muss noch einmal `sendmail -q` durchgeführt werden.

Der Parameter `-k` sorgt dafür, dass die Mails auf dem fernen POP3-Server nicht nach dem Lesen gelöscht werden. Insbesondere in der Testphase ist das eine hilfreiche Option. `fetchmail` liest nur ungelesene Mails. Sollen alle gelesen werden, verwenden Sie `-a`. Die Option `-L` schreibt die Protokolle in die angegebene Datei hier `fetchmail.log` im Heimatverzeichnis.

[Optionen von fetchmail]L|L Option & Bedeutung

`-k` & Mails auf POP3 Server nicht löschen

`-a` & lese alle Mails, nicht nur die ungelesenen

`-L` Datei & Protokolliere in Datei

Mailserver und Domain

Eine E-Mail Adresse erkennt man leicht an dem markanten @-Zeichen in der Mitte. Links von dem @ befindet sich der Benutzername. Rechts davon befindet sich der Hostname des Computers, oft gefolgt von der Domäne, zu der der Computer gehört. Da aber der Benutzer meistens auch in der Domäne bereits eindeutig ist, können Sie sich die Angabe des Computers sparen, wenn die Domäne selbst festlegt, wie die Nachricht an den Benutzer weitergeleitet wird. Die Adresse `arnold@willemer.de` besagt, dass der User `arnold` in der Domäne `willemer.de` gemeint ist. Die Domäne `willemer.de` hat einen Mailserver, der mittels DNS (Domain Name Service siehe S. dns) festgelegt werden kann und der die E-Mail für die Domäne weiter verteilt.

In der Konfigurationsdatei (siehe S. name2nr) des DNS-Servers, die die Namen auf die IP-Nummern umsetzt, gibt es eine Zeile mit dem Code MX. Diese Zeile bezeichnet den Mailserver der Domäne. Hier der entsprechende Ausschnitt:

```
@      IN SOA      mail.willemer.edu.    root.mail.willemer.edu.
      (
      .....
      )
;      wer sind die zustandigen Mailserver
      IN MX 10 mail.willemer.edu.
```

In diesem Beispiel ist der Computer namens `mail` für die Domäne `willemer.edu` zuständig. Die Zahl hinter MX ist die Priorität. Werden mehrere Server aufgelistet, wird zuerst derjenige mit der kleinsten Nummer ausgewählt. Erst wenn dieser nicht ansprechbar ist, wird die Post an denjenigen mit der nächstkleinsten Nummer gesendet. Dieser wird in regelmäßigen Abständen versuchen, die Mail an den primär zuständigen Mailserver weiterzugeben, auf dem die Anwender ihre Mail abholen.

« [Post sammeln: fetchmail](#) | [E-Mail](#) | [Erstes Beispiel: Interne Firmenmail](#) »

Erstes Beispiel: Interne Firmenmail

Manche Firma braucht nur die Möglichkeit innerhalb der Firma E-Mails auszutauschen. Dazu können Sie leicht auf einem ausgedienten PC ich habe dazu einmal einen 486er mit 8 MByte Hauptspeicher und einer Festplatte mit 500 MByte eingesetzt. Linux installieren. Anschliessend wird für jeden Mitarbeiter ein Benutzerkonto eingerichtet. Die lokale E-Mail-Verteilung zwischen den Usern wird bei jeder Distribution automatisch installiert. Gegebenenfalls müssen Sie anpassen, wie häufig der Austausch der Mail erfolgen soll. Dies können Sie am Parameter `-q` des `sendmail` festmachen.

Die Option `-q` ohne weitere Argumente führt die Mailverteilung genau einmal aus. Wird dagegen eine Zeitangabe als Parameter angegeben, arbeitet `sendmail` im Hintergrund und wird in dem angegebenen Intervall aktiv. Durch die Buchstaben `h` (Stunden), `m` (Minuten) und `s` (Sekunden) wird die Zeit angegeben. Beispielsweise bedeutet `-q 1h30m`, dass `sendmail` alle anderthalb Stunden die Verteilung vornimmt.

Was nun noch benötigt wird, ist ein POP3- oder IMAP-Server. Beides wird bei den Distributionen normalerweise mitgeliefert, aber nicht installiert. Sie brauchen hier nicht zu konfigurieren, die Standardinstallation reicht. Damit ist der Server bereits fertig.

Im Netz muss gewährleistet sein, dass jeder Arbeitsplatz den Mailserver erreichen kann. Auf den Clients muss TCP/IP installiert sein und es wird ein POP3-kompatibler E-Mail-Client benötigt. Da POP3 von fast jedem E-Mail-Client bedient wird, ist die Auswahl riesig. Bei IMAP müssen Sie etwas genauer hinsehen, aber die Chance, dass der Client auch diesen Standard beherrscht, ist groß. Der Client braucht nur wenige Parameter. Zunächst benötigt er den Namen oder die IP-Nummer des Mailservers. Dann den Benutzernamen und das Passwort, mit dem sich der Benutzer auf dem Mailserver einloggen kann. Zuletzt wird eingestellt, in welchem Intervall die Clients den Server nach neu eingetroffener Mail abfragen sollen.

Diese Lösung ist recht einfach installiert und extrem billig. Sie hat natürlich den Nachteil, dass der Mailserver ständig abgefragt wird und dass die E-Mail nicht sekundengenau beim Empfänger auf dem Schreibtisch liegt.

Unterabschnitte

- SMTP per sendmail
- E-Mail einsammeln (fetchmail)
- Automatisieren bei Internetverbindung

Zweites Beispiel: Anbindung an das Internet

Das Ziel dieser Beispielkonfiguration ist eine E-Mail Anbindung ans Internet für eine Linux-Workstation mit mehreren Benutzern. Dabei sollte die E-Mail für alle Benutzer der Maschine zentral geholt und versandt werden. Als Distribution wurde SuSE 7.0 verwendet. Der Provider ist T-Online. Es dürfte aber mit jeder anderen Distribution und jedem anderen Provider ähnlich funktionieren.

SMTP per sendmail

Der erste Schritt ist es, einen funktionierenden `sendmail` zu installieren. Dazu wird `sendmail` so vorkonfiguriert, dass SMTP verfügbar ist. Man kann davon ausgehen, dass das bei heutigen Systemen die Standardkonfiguration ist. Anschließend wird die `/etc/sendmail.cf` an einigen Stellen angepasst. In der `sendmail.cf` werden die Stichworte `DeliveryMode`, `^DS` und `^DM` gesucht. Das Dach vor dem D bewirkt, dass diese Buchstaben am Anfang der Zeile stehen müssen (siehe S. `regexp`).

```
O DeliveryMode=deferred
# ...
DSsmtprelay.t-online.de
# ...
DMwillemer.de
```

`DeliveryMode=deferred` führt dazu, dass Nachrichten nur noch versandt werden, wenn `sendmail -q` aufgerufen wird.

`DS` beschreibt den Mailserver, an den abgehende Mails versandt werden. In diesem Fall wurde der Server `smtprelay` von T-Online verwendet. Dieser erlaubt, dass ich meine eigene Domain als Absender verwenden kann.

Hinter `DM` steht die Absendeadresse. Dies kann eine komplette Adresse wie `anton@foo.de` sein oder eine Domäne wie `willemer.de`, die dann einfach an den Nutzernamen angehängt wird. So habe ich ein Account `arnold` auf dem Linuxsystem. Beim Senden erzeugt `sendmail` daraus den Absender `arnold@willemer.de`. Da alle anderen Nutzer auf dem Gerät ebenfalls eine `@willemer.de`-Adresse haben, vereinfacht sich die Installation entsprechend.

Der Startaufruf von `sendmail` bei Starten des Systems befindet sich in der Datei `sendmail` in dem Verzeichnis, indem sich die `rc`-Dateien des Systems befinden. Dort sollten die Attribute des Aufrufs von `sendmail` auf `-bd -om` eingestellt werden. Die Option `-bd` bewirkt das Arbeiten des `sendmail` im Hintergrund auf Port 25 unter dem SMTP. Vor allem wurde hier die Option `-q30m` entfernt, die ansonsten `sendmail` alle 30 Minuten aktiviert hätte.

Zum Test wird mit `elm` oder `mail` eine E-Mail an eine eigene Internetadresse geschrieben. Mit `mailq` sollte die Nachricht anschließend sichtbar sein. Jetzt wird die Verbindung geöffnet und mit `sendmail -q` das Senden angestoßen. Besitzt der Provider einen Webmailer können Sie schnell mit einem Browser prüfen, ob die Mail einwandfrei versendet wurde.

E-Mail einsammeln (fetchmail)

Mit dem Programm `fetchmail` werden drei Mailserver ausgelesen und die dort gefundenen Nachrichten ins lokale E-Mailsystem per SMTP eingespeist. Konfiguriert wird das Programm über die Datei `~/fetchmailrc` des Benutzers `root`. Darin stehen Einträge wie diese:

```
poll pop3.t-online.de protocol POP3 user willemer password . is arnold
poll pop3.web.de protocol POP3 user arnold.willemer password SagIchNicht is arnold
poll pop.gmx.net protocol POP3 user andrea.willemer password VergissEs is andrea
```

Anschließend sollte der folgende Aufruf alle Mailserver abfragen und die E-Mails ins eigene Mailsystem stellen.

```
fetchmail -L /var/log/fetchmail
```

Die E-Mails werden nach dem nächsten Aufruf von `sendmail -q` in die lokalen Briefkästen verteilt

Automatisieren bei Internetverbindung

Bei jedem Aufbau einer Wählverbindung in das Internet wird das Skript `/etc/ppp/ip-up` aufgerufen. Sie können an das Ende einfach folgende Befehle anhängen und es werden bei jedem Verbindungsaufbau die E-Mails zwischen lokalem System und Internet getauscht.

```
echo "Post holen..." >/dev/xconsole
fetchmail >/dev/xconsole
echo "Post senden..." >/dev/xconsole
sendmail -q
echo "fertig!" >/dev/xconsole
```

« [Erstes Beispiel: Interne Firmenmail](#) | [E-Mail](#) | [Newsgroups](#) »

Newsgroups

In den Anfangszeiten des Internets waren die News neben der E-Mail der wichtigste Dienst. Man kann sich Newsgroups als Diskussionsgruppen vorstellen. Ein Newsserver bietet mehrere Gruppen an, die hierarchisch geordnet sind. Eine solche Gruppe heißt beispielsweise

de.comp.os.unix.programming

Das de steht für eine deutschsprachige Gruppe. comp zeigt an, dass es um Computer geht. os bedeutet operating system, also Betriebssystem. unix dürfte klar sein und programming bezieht sich auf das Programmieren. In dieser Gruppe ist das Thema die Programmierung unter UNIX und die Sprache ist deutsch. Sie können per SMTP eine ganz gewöhnliche E-Mail an diese Gruppe senden. Diese wird dann dort gespeichert und ist ab sofort öffentlich. Jeder kann diese Gruppe »abonnieren«. Dazu brauchen Sie einen Newsreader, der alle E-Mails der Gruppe vom Newsserver anfordert, die seit der letzten Abfrage hineingestellt wurden. Auf die Nachrichten können Sie wiederum per E-Mail antworten, so dass eine virtuelle Diskussionsrunde entsteht.

In Zeiten des WWW scheinen sie nur noch ein Forum für Insider zu sein. Die News sind aber nach wie vor der Treffpunkt der Experten und bieten durch die Möglichkeit, Fragen zu stellen, eine Informationsquelle erster Klasse. Damit nicht ständig die gleichen Fragen beantwortet werden müssen, erstellen die meisten Newsgroups eine FAQ (frequently asked questions), die oft einmal monatlich in der Gruppe veröffentlicht wird.

Auch im Intranet einer Firma kann ein Newsserver hilfreich sein. Sie können beispielsweise für Projekte oder Kunden eine Gruppe anlegen, die von allen Mitarbeitern als zentraler Informationspool verwendet werden kann. Jede Absprache mit dem Kunden, jeder Vorgang kann schnell für alle sichtbar protokolliert werden. Da Newsgroups hierarchisch gegliedert werden können, können mehrere Projekte unter einem Kunden angelegt werden. Um an die News zu kommen, benötigen Sie einen Newsreader, der auf allen Plattformen leicht kostenlos zu bekommen ist. Viele E-Mailprogramme beherrschen auch den Umgang mit News.

Unterabschnitte

- [News lesen](#)
- [Installation des Newsservers inn](#)
 - [Der User news](#)
 - [Konfigurationsdateien](#)
- [Beispiel: Newsserver zur Projektverwaltung](#)
- [Gruppen anlegen](#)
 - [Regelmäßige Arbeiten](#)
- [Verbindung nach außen](#)
- [Newsgroups saugen](#)

- Newsgruppenliste vom Internet-Newsserver
 - Zugriffsrechte auf den News-Server
 - NNTP Protokollbeschreibung
 - Clientanfragen
 - Serverantworten
 - Das Senden von Artikeln
 - Threadverfolgung
-

« Zweites Beispiel: Anbindung an | **Netzwerk** | News lesen »

News lesen

Es gibt zwei grundsätzlich unterschiedliche Herangehensweisen, die News von einem Newsserver zu lesen. In beiden Fällen benötigen Sie zunächst einen so genannten Newsreader. Im ersten Fall installieren Sie ihn in hergebrachter Weise, wie Sie das unter MS Windows oder Mac auch täten. Wenn Sie die News lesen wollen, starten Sie auf dem Client den Austausch der News mit dem Server. Als zweite Möglichkeit bietet es sich unter UNIX an, einen Newsserver zu installieren, der die gewünschten Newsgroups vom Newsserver saugt und lokal für den eigenen Rechner oder das Intranet anbietet. Sobald zwei oder mehr Personen News lesen, kann sich diese Vorgehensweise schon lohnen.

Es gibt bei den Newsreadern zwei Kategorien. Die erste geht davon aus, dass der Benutzer ständig zum Newsserver verbunden ist. Wählt er eine Gruppe, beginnt in diesem Augenblick der Client, die Titelzeilen der Nachrichten zu lesen. Wählt der Benutzer eine der Titelzeilen an, wird erst dann der Inhalt der Nachricht geholt. Diese Kategorie nennt man Onlinereader. Der Offlinereader lädt die gewünschten Gruppen am Stück, wenn er mit dem Server verbunden ist und speichert sie lokal. Es gibt auch Varianten, die nur die Titel lesen, die man dann zum Download der eigentlichen Nachrichten markiert.

Zum Einrichten eines Newsreaders brauchen Sie die Adresse des Newsservers und die Adresse des SMTP-Servers, da man eigene News als E-Mail an die Gruppe schreibt und dazu den E-Mail-Briefkasten verwendet.

Unterabschnitte

- [Der User news](#)
- [Konfigurationsdateien](#)

Installation des Newsservers inn

Es gibt derzeit zwei wichtige Newsserver: cnews und inn. cnews ist etwas älter und inn gilt als moderner und leichter bedienbar. Hier soll inn beschrieben werden. Die Konfiguration und Arbeit mit cnews ist an vielen Stellen ähnlich.

Der Newsserver inn verwendet die folgenden Verzeichnisse:

[Verzeichnisse des Newsservers]L|L Verzeichnis & Inhalt
/etc/news & Konfiguration
/var/lib/news & Abgleichsdaten mit dem fremden Newsserver
/var/spool/news & die Ablage der Newsgroup Inhalte
/var/log/news & Logdateien.

Der User news

Für die Arbeiten im Newsbereich sollten Sie einen User namens news anlegen. Dieser User gehört in die ebenfalls anzulegende Gruppe news. Die Wartungsarbeiten werden zwar meist vom Administrator durchgeführt, aber die im Hintergrund laufenden Prozesse für den Newsserver müssen nicht privilegiert ablaufen.

Die meisten Arbeiten sollten Sie als User news durchführen. Da man diesem User ungern ein eigenes Kennwort gibt, loggen Sie sich zunächst als root mit entsprechendem Passwort ein und wechseln dann per su (siehe S. su) nach news. Für diesen Schritt brauchen Sie dann kein Passwort mehr, da Sie ja als root su aufrufen.

```
arnold@silver> su -  
Password:  
silver # su - news  
news@silver>
```

Entsprechend können Sie in der Datei /etc/passwd bzw. in der Datei /etc/shadow einfach ein x anstelle des verschlüsselten Passwortes eintragen, um einen direkten Zugang von außen zu verhindern.

Konfigurationsdateien

Zunächst werden in der Datei inn.conf die Basisinformationen abgelegt. Diese Datei ist nach der Installation bereits weitgehend korrekt vorkonfiguriert. Drei Parameter sollten Sie anpassen:

```
# /etc/news/inn.conf
organization:  Sgt. Pepper's Lonely Hearts Club Band
server:       news.willemer.edu
hiscachesize: 256
```

Hinter organization nennt man normalerweise den Firmennamen. Unter server steht der Rechnername des Newsservers inklusive seiner Domäne. Aus Dokumentationsgründen ist es durchaus sinnvoll, den Hostnamen news zu verwenden und diesen dann als Nickname in der Datei /etc/hosts oder im DNS dem eigentlichen Rechner zuzuordnen. Der Parameter hiscachesize bestimmt die Größe des Speichers in KB für das Caching. Bei Newsservern, die sehr gefragt sind, kann die Erhöhung zu einer schnelleren Verarbeitung führen.

Die Datei incoming.conf legt fest, welche fremden Newsserver berechtigt sind, an diesen Newsserver Artikel weiterzuleiten. Für die Aufgabenstellung eines Intranetnewsservers ist beispielsweise nicht gewünscht, dass fremde Rechner Artikel einstellen. Allerdings muss es dem lokalen Rechner selbst erlaubt sein. Entsprechend steht hier nur ein Eintrag:

```
# /etc/news/incoming.conf
peer localhost {
    hostname: localhost
}
```

Artikel in Newsgroups haben normalerweise ein flüchtiges Dasein. Je nachdem, welchen Umfang die durchlaufenden Nachrichten haben, ist die Festplatte schnell von Nachrichten überlaufen, wenn sie nicht nach gewissen Kriterien wieder aufgeräumt werden. Dies steuert die Datei expire.ctl im Verzeichnis /etc/news. Jede Zeile definiert, in welche Gruppen Nachrichten wie lange bleiben können, bevor sie gelöscht werden.

Der erste Eintrag betrifft die Dauer der History-Einträge und beginnt mit /remember/:. Es folgt die Anzahl der Tage, die der Eintrag bleibt.

Alle anderen Einträge beginnen mit dem Namen der betroffenen Newsgroups. Sie können Regeln für mehrere Newsgroups definieren, indem Sie den Stern als Platzhalter verwenden. Dann folgt eine Unterscheidung nach moderierten und unmoderierten Gruppen. Moderierte Gruppen haben einen Moderator, der jede eingehende Nachricht erst prüft, bevor er sie öffentlich in die Newsgroup stellt. Damit sollen bei sensiblen Themen unpassende Beiträge verhindert werden. und schließlich drei Zeitangaben. Sie haben damit zu tun, wann eine Nachricht ausläuft. Manche Nachrichten haben einen Eintrag im Header, wann sie auslaufen. Die erste Zeitangabe sagt, wie lange eine Nachricht mindestens erhalten bleiben sollte. Die dritte Zeitangabe sagt aus, wann sie spätestens gelöscht werden soll. Der zweite Eintrag besagt, wann eine Nachricht gelöscht wird, wenn sie keine eigenen Angaben darüber macht, wann sie gelöscht werden möchte. Die vorgegebene expire.ctl hat den folgenden Inhalt:

```
# /etc/news/expire.ctl
/remember/:14
*:A:1:10:never
```

Diese Konfiguration besagt, dass Einträge in der History 14 Tage bleiben. Alle Nachrichten in allen Gruppen bleiben mindestens einen Tag, werden normalerweise nach 10 Tagen gelöscht. Der Auslaufvermerk im Header kann aber eine beliebige Dauer vorgeben. Sollen die Nachrichten keinesfalls gelöscht werden, dann sollte der zweite Zeitwert von 10 auf never gesetzt werden. Das ist

beispielsweise erforderlich, wenn der Newsserver zur firmeninternen Projektdokumentation im Intranet eingesetzt werden soll.

Die Angaben in zweiten Feld können folgende Zeichen haben:

[Gruppenkennzeichnung]C|L Zeichen & Bedeutung

M & moderierte Gruppen

U & unmoderierte Gruppen

A & Alle Gruppen

Über die Datei expire.ctl gibt es eine ausführliche Manpage.

« [News lesen](#) | [Newsgroups](#) | [Beispiel: Newsserver zur Projektverwaltung](#) »

Beispiel: Newsserver zur Projektverwaltung

Der erste Schritt der Konfiguration soll am Beispiel eines Newsservers für das Intranet einer Firma demonstriert werden. Darin sollen Diskussionsgruppen zu Kunden und Projekten eingerichtet werden. Hier können Absprachen und Abläufe protokolliert werden. Die Bedienung ist genauso einfach wie das Schreiben einer E-Mail. Bei einer solchen Anwendung ist keine Anbindung an andere Newsserver vorgesehen, und auch gar nicht gewünscht, da die Diskussionen ja firmenintern sind und nicht nach außen dringen sollen.

Zunächst wird der inn-Dämon von Hand gestartet. Da der Dämon auf den well known Port 119 zugreifen wird, darf er nur vom Superuser gestartet werden. An dieser Stelle wird das Startskript verwendet, das auch benutzt wird, wenn die Maschine bootet.

```
gaston# /etc/init.d/inn start
```

Sie können im ersten Schritt prüfen, ob der `inn` in der Prozessliste auftaucht. Ist dieser gestartet, können Sie nachsehen, ob der `inn` Anfragen an Port 119 beantwortet. Das einfachste Testtool ist `telnet`, dem Sie als Parameter den Port 119 mitgeben.

```
gaston> telnet localhost 119
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
200 gaston.willemer.edu InterNetNews server INN 2.2.3 18-Jul-2000 ready
quit
205 .
Connection closed by foreign host.
gaston>
```

Der Befehl `quit` sorgte dafür, dass die Verbindung zwischen `telnet` und `inn` wieder aufgelöst wurde. Gibt es eine andere Meldung, insbesondere eine Meldung »Connection refused« ist der Server nicht gestartet. Vermutlich stimmt eine Konfigurationsdatei nicht. Ein Blick in die Protokolldatei des `syslog`-Dämons (siehe S. `syslog`) gibt weitere Informationen über die Ursache des Problems.

Mit einem Newsclient können Sie den Server ansprechen, indem Sie den Rechnernamen oder einfach »localhost« als Newsserver angeben. Anschließend versuchen Sie, Gruppen zu abonnieren. Dabei wird der Client die Gruppenliste holen. Die besteht zu Anfang aus folgenden Gruppen:

```
control
control.cancel
junk
```

Diese Gruppen sind zum Betrieb des Newsservers `inn` erforderlich.

Unterabschnitte

- [Regelmäßige Arbeiten](#)

Gruppen anlegen

Die Administration des Newsgroupservers erfolgt durch das Senden von Kontrollnachrichten. Das zentrale Programm heißt `ctlinnd`.

Der erste Parameter von `ctlinnd` ist ein Kommando an den Newsserver. Zum Einrichten einer Newsgroup lautet er `newgroup`. Es folgt der Name der Gruppe. Gruppen sind hierarchisch angeordnet und durch Punkte getrennt. Danach folgt ein `y` oder `n` dafür, ob lokale Benutzer in die Gruppe schreiben dürfen. Am Ende der Zeile steht schließlich der Erzeuger der Gruppe. Die folgenden Befehle erstellen ein Grundgerüst für den Kunden Maier, bei dem ein Netzwerk mit Workstations und einem Server erstellt werden soll. Beim Server werden zusätzlich noch Ausfälle in der Untergruppe `breakdown` protokolliert.

```
gaston> ctlinnd newgroup maier y news
gaston> ctlinnd newgroup maier.netzwerk y news
gaston> ctlinnd newgroup maier.workstations y news
gaston> ctlinnd newgroup maier.server y news
gaston> ctlinnd newgroup maier.server.breakdown y news
```

Um Gruppen wieder zu löschen, verwenden Sie das Kommando `rmgroup`. Um die Schreibberechtigungen zu korrigieren, können Sie `changegroup` verwenden oder noch einmal das Kommando `newgroup` mit den richtigen Parametern wiederholen. `inn` merkt, dass die Gruppe schon existiert und führt das Kommandos als Änderung aus.

Für die Schreibberechtigung der Gruppe steht ein `y` für die Erlaubnis zu schreiben. `n` verbietet das Schreiben auf dem lokalen Rechner und `x` unterbindet jegliches Schreiben in dieser Gruppe, auch von außen. Eine genauere Beschreibung finden Sie in der Manpage der Datei `active`.

Regelmäßige Arbeiten

In der `crontab` des Users `news` sollte ein Eintrag für `news.daily` stehen. Ansonsten sendet das news-System dem Postmaster eine Warnmeldung per Mail.

```
# taegliche wartungsarbeiten in der crontab
07 01 * * * /usr/lib/news/bin/news.daily
```

Diese Zeile sagt aus, dass `news.daily` immer nachts um 1:07 startet. Es wird in diesem Zuge eine E-Mail an den Newsmaster mit einer Statistik gesendet.

Verbindung nach außen

In der nachfolgenden Konfiguration soll ein Newsserver nicht als internes Diskussionsforum dienen, sondern an einen Newsserver im Internet angebunden werden, so dass Gruppen von dort in den lokalen Server übernommen werden und Beiträge der lokalen Benutzer auch ins Internet gehen.

Zunächst werden im Verzeichnis `/var/lib/news` die Dateien `active` und `newsgroups` um die Gruppe `de.test` ergänzt. Die Datei `active` enthält den Namen jeder Gruppe, die vom Newsserver bearbeitet wird. Hinter dem Gruppennamen befindet sich die höchste Artikelnummer, die in dieser Gruppe bisher gelesen wurde. Die folgende Nummer ist die niedrigste Artikelnummer. Da Artikel auch zurückgezogen worden sein können, ist die Differenz nicht zwingend die Anzahl der vorhandenen Artikel. Zuletzt steht eine Option, die besagt, ob in diese Gruppe Nachrichten eingestellt werden dürfen. Für die Gruppe `de.test` wird folgende Zeile eingetragen:

```
de.test 0000000001 0000000001 y
```

In der Datei `newsgroups` wird die Gruppe mit einem erläuterndem Text aufgeführt.

```
de.test          Testing group
```

Die Gruppe `de.test` ist auf den meisten Newsservern verfügbar, die deutsche Newsgroups anbieten. Hier stellen alle diejenigen kurze Tests ab, die prüfen wollen, ob ihre Newssoftware richtig funktioniert. Zum Test, ob die News richtig gelesen werden, eignet sich diese Gruppe deswegen sehr, weil dort ständig ein gewisser Verkehr herrscht und die Nachrichten nicht groß sind. Dazu kommt, dass Sie bereits in der richtigen Gruppe sind, wenn Sie später prüfen wollen, ob auch das Schreiben funktioniert.

Nachdem die Gruppen festgelegt sind, wird nun der Newsserver festgelegt, von dem die Nachrichten geholt werden und an den die lokal abgestellten Nachrichten weitergeleitet werden.

In der Datei `/etc/news/newsfeeds` wird konfiguriert, von welchem Newsserver der lokale Newsserver gefüttert werden darf. Die meisten Einträge lassen Sie am besten unberührt. Interessant ist eine Zeile, die Sie hinzufügen müssen. Für den Newsserver von T-Online steht dort beispielsweise:

```
# /etc/news/newsfeeds
newsserv/news.t-online.de:*:Ap,Tf,wnm:newsserv
```

Der Begriff `newsserv` ist frei gewählt und dient als Kennung für diesen Eintrag. Hinter dem Schrägstrich erscheinen kommasepariert die Newsserver, von denen Beiträge eingelesen wurden. So wird verhindert, dass die Beiträge des einen Newsservers wieder an den anderen hinausgehen. Hinter dem ersten Doppelpunkt erscheint die Liste der Newsgroups. Der Stern sagt aus, dass alle denkbaren Gruppen des Servers erlaubt sind.

Hinter dem Doppelpunkt erscheinen Flags, die durch Kommata getrennt sind. Wer Newsgroups von einem Newsserver herunter lädt, kann die hier stehenden Parameter verwenden. Weitere Parameter finden sich in der Manpage von `newsfeeds`. Die hier genannten Flags bedeuten:

- [Ap] Vor dem Senden soll der Name des Newsservers nicht geprüft werden.
- [Tf] Das Senden erfolgt in Form einer Datei.
- [Wnm] Innerhalb der Datei werden die Artikel anhand ihrer Artikelnummer und ihres Namens unterhalb des Spoolverzeichnisses gekennzeichnet.

Als letztes Feld wird die Kennung vom Anfang wiederholt.

Falls der Newsserver bereits lief, muss die Konfiguration aktualisiert werden. Dazu kann wieder das Administrationstool `ctlinnd` verwendet werden. Mit dem Kommando `reload newsfeeds` wird die Konfigurationsdatei erneut eingelesen.

```
ctlinnd reload newsfeeds do
```

« [Gruppen anlegen](#) | [Newsgroups](#) | [Newsgroups saugen](#) »

Unterabschnitte

- [Newsgruppenliste vom Internet-Newsserver](#)
- [Zugriffsrechte auf den News-Server](#)

Newsgruppen saugen

Im nächsten Schritt werden Daten von einem Newsserver heruntergeladen. Dazu gibt es das Hilfsprogramm `suck`, das Nachrichten von einem Server lädt und in das lokale Newssystem einspeist. Die Konfigurationsdatei heißt `sucknewsrsc` und muss im aktuellen Verzeichnis stehen. Für den ersten Versuch wird nur eine Newsgroup verwendet, nämlich `de.test`. Der Eintrag in die `sucknewsrsc`-Datei lautet:

```
de.test 1 100
```

Hinter dem Gruppennamen steht die Artikelnummer, ab der man aufsetzen möchte. Da nicht bekannt ist, welche Artikelnummer derzeit aktuell ist, verwenden Sie zunächst die 1. Danach wird beschränkt, wieviele Artikel heruntergezogen werden sollen. 100 Artikel sind mehr als genug und sind in wenigen Sekunden geladen. Diesen dritten Parameter, der die Mengenbegrenzung darstellt, können Sie auch weggelassen. Dann werden alle existierenden Nachrichten geladen.

Der Aufruf, um die lokalen Newsgroups eines Rechners mit den News des Newsservers `news.t-online.de` zu füllen, lautet:

```
suck news.t-online.de -bp -h1 localhost
```

Der Parameter `-h1 localhost` bedeutet, dass die Newsgroups in den eigenen Server gespeist werden. Der Parameter `-bp` bedeutet batchpost und bewirkt, dass zunächst alle Nachrichten am Stück in eine lokale Datei geschrieben werden und von dort im Block in den lokalen Newsserver geschoben werden.

Während seines Laufs benennt `suck` die Datei `sucknewsrsc` in `suck.newrc` um. Die Nummern der als nächstes zu holenden Nachrichten werden darin aktualisiert. Vor dem nächsten Start muss also `suck.newrc` nach `sucknewsrsc` umkopiert werden, damit die gleichen Nachrichten nicht wieder geholt werden. Damit `suck` dies selbst tut, verwenden Sie die Option `-c`.

Sie können auch die Datei `active` vom Newsserver als Basis für die zu ladenden Newsgroups verwenden. Dies erreichen Sie mit dem Parameter `-A`. Allerdings sollten Sie damit erst eine Weile experimentieren. In bestimmten Fällen scheint dabei die Datei `active` durcheinander zu geraten.

Newsgruppenliste vom Internet-Newsserver

Die Dateien `active` und `newsgroups` im Verzeichnis `/var/lib/news` können Sie sich vom Newsserver herunterladen. Bevor Sie dies tun, ist es sinnvoll, die bisherigen Dateien `active` und `newsgroups` zu

sichern.

```
cd /var/lib/news
getlist -h news.t-online.de active      > active
getlist -h news.t-online.de newsgroups > newsgroups
```

Jeder Aufruf `getlist` zieht jeweils etwa 1 MByte über die Leitung. Je nach Geschwindigkeit der Internetanbindung brauchen Sie also etwas Geduld.

Zugriffsrechte auf den News-Server

Die Zugriffsbeschränkungen auf den News-Server werden in der Datei `/etc/news/nntp.access` definiert. In jeder Zeile wird beschrieben, welche Rechner welche Rechte auf dem Server haben. In der ersten Spalte wird die Rechnergruppe, für die diese Zeile gilt. Die Rechneridentität wird durch eine Maske beschrieben. Bei unscharfer Beschreibung gilt die genaueste Beschreibung, die auf den Host zutrifft. Rechner können auf folgende Arten beschrieben werden.

[Berechtigte]L|L Ziel & Bezeichnungsweise
Hostname & wird durch IP-Reverse Auflösung bestimmt
IP-Nummer & im dotted decimal Format
Domain & beispielsweise `*.willemer.edu`
Netzwerkname & nach `/etc/networks`
default & gilt für alle Rechner

Die zweite Spalte beschreibt die Berechtigungen bzgl. des Austausches von Nachrichten.

[Rechte]L|L Recht & Berechtigt zu ...
read & Artikel holen
xfer & Artikel in den Server hineinschieben
both & sowohl read als auch xfer
no & gar nichts

Die dritte Spalte ist für das »Posten« von Artikeln. Darunter versteht man das Senden von Nachrichten an einer Gruppe. Dies betrifft einen Client der lokalen Maschine. Eine gepostete Nachricht erhält eine eindeutige Nummer mit dem Rechnernamen, von dem sie stammt.

[Berechtigung zu Posten]L|L Parameter & Berechtigung
post & Artikel »posten«
no & keine Artikelerstellung

« [Verbindung nach außen](#) | [Newsgroups](#) | [NNTP Protokollbeschreibung](#) »

Unterabschnitte

- [Clientanfragen](#)
 - [Serverantworten](#)
 - [Das Senden von Artikeln](#)
 - [Threadverfolgung](#)
-

NNTP Protokollbeschreibung

Das NNTP (Network News Transfer Protocol) ist durch RFC 977 definiert. Die folgende Beschreibung bietet einen Blick hinter die Kulissen eines Internetprotokolls, das durchaus typisch ist. Diese Informationen sind in erster Linie für Entwickler von Programmen interessant, die sich im Protokollumfeld bewegen. Aber es ist auch für den Administrator hilfreich zu wissen, wie die Kommunikation abläuft, um beispielsweise mit `telnet` das Fehlverhalten von Protokollen herausfinden zu können.

Das Protokoll wird durch den Austausch von Kommandos und Antworten in Textform bestimmt. Die rein textuelle Übermittlung der Daten und Kommandos erspart die typischen Netzprobleme mit Binärdaten wie die unterschiedliche interne Zahlendarstellung verschiedener Maschinen.

Ein Newsserver ist im Gegensatz zu einem Webserver nicht statuslos. Das bedeutet, er kann aufeinander aufbauende Anfragen in einen Zusammenhang stellen. Darum ist eine Anmeldung und auch eine Abmeldung erforderlich. Es wird ein so genannter Messagepointer geführt, mit dem sich der Server die zuletzt angesprochenen Nachrichten merkt.

Der Client muss die Kommunikation aufnehmen und sollte in der Lage sein, alle denkbaren Antworten des Servers zumindest soweit zu bearbeiten, dass die Kommunikation nicht blockiert.

Clientanfragen

Vom Client werden die Anfragen an den Server gesandt. Die wichtigsten Kommandos des Clients sind hier aufgeführt. Sie reichen bereits aus, einen einfachen Newsreader zu schreiben.

- [LIST]
Der Client fordert eine Liste der Gruppen an. Das Format entspricht dem der Datei `active`. Auf die Aufforderung `LIST` sendet der Server beispielsweise folgende Ausgabe:

```
215
control 0000000000 0000000000 y
control.cancel 0000000000 0000000000 y
junk 0000000000 0000000000 y
.
```

- [NEWGROUPS Datum Uhrzeit]
Es werden die Gruppen angefordert, die seit dem angegebenen Zeitpunkt hinzugekommen sind. Das Datum ist im Format `YYMMDD`. Da das Jahr zweistellig ist, gilt das Jahrhundert, das näher an der Jahreszahl ist. Die Uhrzeit hat das

Format HHMMSS.

- [NEWNEWS Gruppenname Datum Uhrzeit]
Auf diese Anforderung wird eine Liste von Artikelkennungen gesendet. Jede Kennung belegt eine Zeile, die mit Carriage Return und Line Feed abgeschlossen wird. Das Ende der Liste wird durch einen einzelnen Punkt in einer Zeile signalisiert.
- [GROUP Gruppenname]
Der Client wechselt zu dieser Gruppe. Als Antwort erhält der Client eine Fehlernummer (siehe unten). Dann folgt die Anzahl der Artikel, dann die kleinste Artikelnummer und schließlich die höchste Artikelnummer. Die Anzahl der Artikel ergibt sich nicht einfach der Differenz der Artikelnummern, weil diese durchaus Lücken enthalten können. Zuletzt erscheint zur Bestätigung noch einmal der Gruppenname.

Üblicherweise wird der Client anschließend die niedrigste Nummer als Argument für das Kommando ARTICLE oder STAT verwendet.

- [ARTICLE Artikelkennung]
Als Antwort auf diesen Anforderung wird der Artikelkopf (wie bei HEADER), eine leere Zeile und dann der Artikeltext (wie bei BODY) angezeigt.
- [HEAD]
Zeigt den Kopf der aktuellen Nachricht. Die Übertragung endet mit einer Zeile, die nur einen einzelnen Punkt enthält.
- [BODY]
Zeigt den Inhalt der aktuellen Nachricht. Die Übertragung endet mit einer Zeile, die nur einen einzelnen Punkt enthält.
- [STAT Artikelkennung]
Wie ARTICLE, allerdings wird der Artikel nicht übertragen. Es wird also nur auf die angegebene Artikelkennung positioniert.
- [NEXT]
Fordert den nächsten Artikel an. Als Ergebnis gibt es eine Fehlernummer (siehe unten) gefolgt von der Artikelkennung. Um den Artikel zu betrachten, muss HEAD oder BODY verwendet werden.
- [QUIT]
Beendet die Sitzung.

Serverantworten

Auf jede Anfrage wird der Server zun"achst mit einem dreistelligen Zahlencode reagieren. Erst hinter dieser Zahl steht die eigentliche Antwort. Dabei bedeutet:

[Fehlercode]L|L Zahlencode & Bedeutung

1xx & Informativ

2xx & Kommando ok

3xx & Kommando soweit ok, sende den Rest!

4xx & Kommando war korrekt, konnte aber nicht ausgef"uhrt werden.

5xx & Kommando unbekannt oder Fehler

Die n"achste Stelle sagt etwas "uber die Kategorie:

[Fehlerkategorie]L|L Zahlencode & Bedeutung

x0x & Verbindung, Setup und sonstige Nachrichten

- x1x & Newsgroupauswahl
- x2x & Artikelauswahl
- x3x & Distributionsfunktionen
- x4x & Senden von Artikeln
- x8x & Erweiterungen, die nicht standardisiert sind.
- x9x & Debug-Ausgaben

Das Senden von Artikeln

Nach dem Senden von POST wird der Server zun"achst mitteilen, ob ihm das Senden behagt. Der Antwort-Code kann lauten:

[Fehlerursache]L|L Zahlencode & Bedeutung

240 & Der gesendete Artikel ist ok

340 & Sende Artikel. Endekennung ist eine Zeile mit nur einem Punkt.

440 & Posten nicht erlaubt

441 & Posten fehlgeschlagen

Anschließend sendet der Client Zeile für Zeile seine Nachricht, die der RFC 850 f"ur das Format einer Nachricht entsprechen muss. Speziell f"ur die Usenet News Artikel gilt die RFC 1036.

Threadverfolgung

Threads sind Diskussionsb"äume, die durch Antworten auf Artikel entstehen. Im Artikel wird durch Belegen des Feldes References: die Message-ID abgelegt, auf die der Artikel sich bezieht. Es gibt also R"uckw"arts- aber keine Vorw"artsbez"uge.

« [Newsgroups saugen](#) | [Newsgroups](#) | Jeder Rechner ein eigener »

Jeder Rechner ein eigener Webserver

***Linux is like a wigwam. No Gates no Windows
but always an apache inside Zitat unbekannter
Herkunft***

Das Programmpaket Apache ist ein Open Source Projekt und gilt als der meistverwendete Webserver im Internet. Den passenden Client nennt man Browser, die bekanntesten sind sicher der Netscape Navigator und der Microsoft Internet Explorer im MS Windows-Umfeld. Ursprünglich diente das Web der Vernetzung von Texten, die Wissenschaftler zur Verfügung stellten und in denen sie mit einem Link auf die Ergebnisse von Kollegen verwiesen. Inzwischen ist das Web allerdings zu einer Welt der bunten Bilder und der Animationen mutiert.

Ein Webserver ist ein Prozess der Anfragen nach dem Protokoll HTTP beantwortet. HTTP ist die Abkürzung von Hypertext Transfer Protocol und ist in RFC 1945 für die Version 1.0 definiert. HTTP ist wohl der Dienst, der im Internet am populärsten ist. Apache ist ein HTTP-Dämon und darum heißt der eigentliche Prozess httpd.

Die Konfiguration des Servers erfolgt in der Datei httpd.conf. Darin sind alle Einstellungen zum Webserver zentral abgestellt. Hier kann auch freigeschaltet werden, dass einzelne Verzeichnisse durch eine Datei .htaccess konfigurierbar sind. Die Datei .htaccess folgt der Datei http.conf in der Syntax. Viele Provider erlauben ihren Kunden, eigene .htaccess-Dateien zu verwenden. Damit können Sie die Kenntnisse der Konfiguration des Apachen auch für die eigenen Webseiten anwenden, selbst wenn Sie bei einem fremden Provider keinen Zugriff auf die Datei httpd.conf haben (siehe S. htaccess).

Unterabschnitte

- [Hypertext und HTML](#)
 - [Gestaltungselemente](#)
 - [Formulare](#)
- [Start des Servers](#)
- [Die Konfigurationsdatei httpd.conf](#)
 - [Directory-Einstellungen](#)
- [Privatadministration per .htaccess](#)
- [Kommunikation per HTTP](#)
 - [Serverantworten](#)
- [Virtuelles Hosting](#)
- [CGI: der Server schlägt zurück](#)
 - [Daten an den Server senden](#)

Unterabschnitte

- [Gestaltungselemente](#)
- [Formulare](#)

Hypertext und HTML

Ein Hypertext ist ein Text mit aktiven Verweisstellen auf andere Texte. Im Web werden die Verweise Links genannt. So ist es möglich, Texte zu schreiben, die eine Materie auf hohem Niveau behandeln. Anfänger können durch die Verweise auf die erforderlichen Erläuterungen und Definitionen geführt werden, während der Experte seine Informationen komprimiert erhält. Hypertexte eignen sich auch ideal, um Informationen abzulegen, die stark miteinander verknüpft ist.

Die Sprache HTML (Hypertext Markup Language) ist sehr einfach zu lernen. Um die Beispiele besser zu verstehen, werden hier einige ausgewählte Grundlagen vorgestellt. HTML-Dateien können mit einem gewöhnlichen Editor erstellt werden und haben neben dem eigentlichen Text Darstellungsbefehle, die Tag (engl. Etikett) genannt werden. Ein Tag ist in spitzen Klammern eingeschlossen. In den meisten Fällen gilt ein Tag für einen Bereich. Der Geltungsbereich wird durch das Ende-Tag begrenzt, dass Sie daran erkennen, dass ein Schrägstrich hinter der geöffneten, spitzen Klammer steht. Die Startseite einer Website heißt normalerweise index.html oder index.htm. Sie hat folgenden Inhalt:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<H1>Große Überschrift</H1>
```

Schauen Sie sich diesen `Hinweis` an.

```
</BODY>
</HTML>
```

Die Tags HTML und BODY rahmen den Körper einer Webseite ein. Der Abschnitt HEAD enthält Informationen, die beispielsweise Suchmaschinen auswerten, und auch den Titel der Webseite. H1 ist eine Überschrift (engl. header). Es gibt noch H2, H3, H4 und H5, die immer kleiner werdende Überschriften erzeugen. Das Tag A dient der Adressierung und besitzt Attribute. Mit HREF (Hyperreferenz) wird auf die angegebene Datei, hier hinweis.html verwiesen. Der Text der zwischen `` und `` wird bei der Darstellung durch den Browser unterstrichen und blau dargestellt. Damit wird signalisiert, dass man durch einen Klick hierauf an eine andere Stelle springt. In diesem Fall führt es zum Laden der Datei hinweis.html auf dem gleichen Server.

Sie müssen noch die Seite hinweis.html erzeugen. Sonst weist der Link von index.html ins Leere. Sie sieht nicht viel anders aus als die Datei index.html.

```
<HTML>
<BODY>
<H1>Großer Hinweis</H1>
```

```
<H4>Kleiner Hinweis</H4>
```

```
Und hier geht es <A HREF="index.html">zur Hauptseite</A>.
```

```
</BODY>
</HTML>
```

Gestaltungselemente

Aller Text, der von dem Tag BODY eingerahmt wird, wird vom Browser einfach ohne Zeilenumbruch hintereinander geschrieben. Um einen Absatz zu erzeugen, wird das Tag <P> eingefügt. Soll nur einen Zeilenumbruch ohne Abstand erreicht werden, verwenden Sie
.

Eine in HTML geschriebene Tabelle sieht auf den ersten Blick etwas kompliziert aus. Das liegt daran, dass sie gleich drei Tags verwendet. Das erste ist TABLE, das die gesamte Tabelle umschließt. TR umschließt jede Tabellenzeile und TD eine Spalte innerhalb einer Zeile. Der Browser übernimmt die korrekte Ausrichtung. Sie brauchen sich also keine Gedanken darüber zu machen, welchen Platz eine Spalte benötigt. Das folgende Beispiel erzeugt eine Tabelle, die noch einmal die Tags einer Tabelle beschreibt:

```
<TABLE BORDER>
<TR><TD> table </TD><TD> Die ganze Tabelle </TD></TR>
<TR><TD> tr   </TD><TD> Eine Spalte      </TD></TR>
<TR><TD> td   </TD><TD> Ein Feld          </TD></TR>
</TABLE>
```

Das Attribut BORDER versieht die Tabelle mit einem Rahmen. Auch ohne Rahmen sind Tabellen wichtige HTML-Elemente. Man benötigt sie, um Texte nebeneinander zu positionieren. Durch geschicktes Verschachteln mehrerer Tabellen können auch kompliziertere Anordnungen realisiert werden.



Formulare

Zur Erstellung von Eingabemasken und Kontrollelementen wird in HTML ein Formular verwendet. Der Inhalt des Formulars wird entweder auf dem Server selbst verarbeitet oder kann per E-Mail versandt werden. Die Verarbeitung der Formularinhalte wird später noch einmal beim Thema CGI auftauchen (siehe S. cgi).

Jedes Formular wird unsichtbar durch das Tag FORM eingerahmt. Im Starttag wird Aktion und Versendungsmethode festgelegt.

```
<FORM ACTION="mailto:arnold@willemer.de" METHOD=post>
...
</FORM>
```

Dies ist ein typisches Beispiel für das Versenden des Formularinhaltes per E-Mail. Der Browser wird versuchen, den Formularinhalt an die genannte E-Mail Adresse zu senden. Andere Aktionen werden unter dem Thema CGI näher erläutert.

Ein Formular wird erst durch die Bestückung mit Kontrollelementen lebendig. Ein Kontrollelement wird mit dem Tag INPUT gekennzeichnet. Mit dem Attribut TYPE wird der Typ gekennzeichnet werden. Das wichtigste Kontrollelement eines Formulars heißt SUBMIT. Es ist der Bestätigungsknopf, der dafür sorgt, dass die Eingaben abgeschickt werden. Mit dem Attribut VALUE kann die Vorbelegung des Eingabeelements erfolgen. Beim SUBMIT ist dies die Beschriftung. Ganz wichtig ist das Attribut NAME, an dem Sie bei der Auswertung die einzelnen Kontrollelemente erkennen können. Also müssen die Namen jedes Kontrollelements für das jeweilige Formular eindeutig sein.

Jedes Formular enthält normalerweise einen Button zum Absenden des Formularinhalts an das CGI-Programm oder die E-Mail. Dieser Button ist vom Typ SUBMIT. Erst wenn der Betrachter diesen Knopf betätigt, wird die ACTION des Formulars aktiv.

```
<FORM>
<INPUT TYPE=SUBMIT VALUE="OK">
</FORM>
```

Mit dem Kontrollelementtyp TEXT wird ein Feld zur Eingabe einer Textzeile erzeugt. Beispiel:

```
<FORM>
welches Auto fahren Sie?
<INPUT TYPE=TEXT NAME="auto" VALUE="Alt aber bezahlt"
  SIZE=20 MAXLENGTH=60>
<TEXTAREA Name="Adresse" ROWS=5 COLS=60>
Ihre Adresse
</TEXTAREA>
<INPUT TYPE=SUBMIT VALUE="OK">
</FORM>
```

Eine Abart des Kontrollelements TEXT ist die TEXTAREA, die mehrere Zeilen Text aufnehmen kann. Man könnte auch von einem simplen Editor reden. Er wird mit dem Tag TEXTAREA eingeleitet. Der folgende Text bis zum Endetag ist die Vorbelegung des Kontrollelements, ähnlich wie bei VALUE=.



Ein zusammengesetztes Element ist der Radiobutton. Es werden mehrere »Knöpfe« unter gleichem Namen definiert. Nur einer von ihnen kann selektiert werden. Die folgende Definition bildet realistisch die möglichen Eigenarten von Menschen ab.

```
<INPUT TYPE="RADIO" NAME="eigenschaft" VALUE="intelligent">
<INPUT TYPE="RADIO" NAME="eigenschaft" VALUE="sportlich">
<INPUT TYPE="RADIO" NAME="eigenschaft" VALUE="schön">
```

Wer nähere und aktuelle Informationen zum Thema HTML braucht, sollte sich mit SelfHTML beschäftigen. Es wird bei vielen Linuxdistributionen mit ausgeliefert. Ansonsten finden Sie es im Internet unter der Adresse:

<http://selfhtml.teamone.de>

« [Jeder Rechner ein eigener](#) | [Jeder Rechner ein eigener](#) | [Start des Servers](#) »

Start des Servers

Eine Standardinstallation ist nicht besonders aufwändig. Die entscheidende Konfigurationsdatei heißt `httpd.conf` und steht im Verzeichnis `/usr/local/apache/conf`, sofern dem Server beim Start nicht mit der Option `-f` eine andere Datei angewiesen wurde. Das lässt sich durch einen Blick in die Prozessliste feststellen, wie unten zu sehen:

```
gaston> ps ax | grep http
  839 ?      S    0:00 /usr/sbin/httpd -f /etc/httpd/httpd.conf
  860 ?      S    0:00 /usr/sbin/httpd -f /etc/httpd/httpd.conf
1247 pts/3  S    0:00 grep http
gaston>
```

Die Datei `httpd.conf` liegt also im Verzeichnis `/etc/httpd`. Alternativ können Sie natürlich auch in den `rc`-Dateien nachsehen, auf welche Weise `httpd` gestartet wird.

Wenn der Server gestartet ist, sollten Sie mit einem Browser bereits erste Seiten abrufen können. Das sind entweder die Testseiten des Apache oder ihre eigenen Seiten, wenn Sie sie an der richtigen Stelle abgelegt haben. Falls Sie lokal einen Browser zur Verfügung haben, starten Sie ihn und geben in der Adresszeile `http://localhost` ein. Sollten Sie einen anderen Computer mit Browser im Netz haben, können Sie den Server auch über das Netz ansprechen und hinter die beiden Schrägstriche den Rechnernamen des Servers oder dessen IP-Nummer setzen.

Unterabschnitte

- [Directory-Einstellungen](#)

Die Konfigurationsdatei httpd.conf

Die Einstellungen in der Datei httpd.conf werden über Schlüsselworte durchgeführt, mit denen die Zeile anfängt. Den Rest der Zeile füllt der Wert. Wie unter UNIX üblich, ist # das Kommentarzeichen. Leere Zeilen werden ignoriert. Die Schlüsselwortzuordnungen in der Datei httpd.conf werden Direktiven genannt. Die globalen Direktiven beschreiben, wie der Server arbeitet. Hier einige Standardeinstellungen für einen nicht unter Last stehenden Server.

```
ServerType standalone
ServerRoot "/usr/local/httpd"
StartServers 1
MaxClients 150
```

Die Direktiven haben folgende Bedeutungen.

- [ServerType] Hier können zwei Werte stehen, standalone und inetd. Im ersten Fall wird der Apache über die rc-Dateien gestartet, im anderen Fall über den Internetdämon. Ein Provider wird hier immer standalone eintragen, damit der Webserver schnell antwortet.
- [ServerRoot] Hier wird festgelegt, wo das Basisverzeichnis für die Daten des Servers liegt. Hierunter befinden sich die Verzeichnisse htdocs für die Dokumente und cgi-bin für die CGI-Dateien. Daneben aber auch Bibliotheken wie beispielsweise die für PHP.
- [StartServers] Anzahl der parallel startenden Serverprozesse. Jeder der Prozesse kann beliebig viele Anfragen parallel bearbeiten. Bei jeder Anfrage wird ein neuer Prozess generiert, der nach der Bearbeitung terminiert.
- [MaxClients] Hier wird begrenzt, wieviele parallele Prozesse gleichzeitig arbeiten dürfen. Diese Konstante ist ein Sicherheitswert, der verhindern soll, dass der Server derart überlastet wird, dass er nicht mehr administrierbar ist.

Die nächsten Direktiven beschreiben die Umgebung des Servers und an welcher Stelle die Dokumente stehen.

```
ServerAdmin root@gaston.willemer.edu
ServerName gaston.willemer.edu
DocumentRoot "/usr/local/httpd/htdocs"
```

- [ServerAdmin] Das ist die E-Mailadresse, an die Probleme weitergeleitet werden.
- [ServerName] Hier ist der Name, der nicht zwingend dem Hostname des Rechners entsprechen muss. Hier könnte also auch www.willemer.edu stehen. Allerdings muss der Name durch /etc/hosts oder DNS bekannt sein, da ansonsten httpd nicht startet.
- [DocumentRoot] Hier wird das Verzeichnis genannt, in dem die eigentlichen HTML-Dateien liegen. Von außen gesehen ist dies das WWW"-Rootverzeichnis des Servers.

Die hinter dem Schlüsselwort DirectoryIndex stehenden Dateinamen werden automatisch geladen, wenn nur der Verzeichnisname angegeben wird. Dabei werden sie der Reihe nach durchgegangen, bis Apache eine Datei diesen Namens findet.

Wenn Sie nun die beiden HTML-Seiten von Seite `html1` eingeben und in das Verzeichnis stellen, das unter `DocumentRoot` in der `httpd.conf` steht, sollten beim nächsten Besuch per Browser die selbsterstellten HTML-Seiten im Netz stehen.

Directory-Einstellungen

Es können Direktiven auf Verzeichnisse angewandt werden. Damit können Rechte für verschiedene Verzeichnisse unterschiedlich geregelt sein. So ist der Zugriff auf das Dokumentverzeichnis normalerweise offener als der auf die CGI-Skripte. Die Abschnitte werden wie durch Tags eingeklammert, wobei hinter dem Einleitungstag der Name des Verzeichnisses steht.

Zunächst wird das Wurzelverzeichnis des Rechners eingestellt. Die `Directory` Einträge gelten für alle Unterverzeichnisse, also wirken sich diese Einstellungen auf die gesamte Maschine aus.

```
<Directory />
    AuthUserFile /etc/httpd/passwd
    AuthGroupFile /etc/httpd/group
    Options -FollowSymLinks +Multiviews
    AllowOverride None
</Directory>
```

Damit werden die Zugriffe soweit wie möglich eingeschränkt. In den folgenden Abschnitten können dann die Zugriffe für spezielle Bereiche gelockert werden. Relevant für den Betrieb des Servers sind die Einstellungen des Verzeichnisses, das oben als `DocumentRoot` eingetragen wurde:

```
<Directory "/usr/local/httpd/htdocs">
    Options Indexes -FollowSymLinks +Includes Multiviews
    AllowOverride All
    Order allow,deny
    Allow from .willemer.edu
</Directory>
```

Die Option `FollowSymLinks` ist normalerweise aus Sicherheitsgründen abgeschaltet. Bei eingeschalteter Option kann mit einem symbolischen Link auf beliebige Verzeichnisse des Rechners verwiesen werden. Das ist praktisch, wenn man Webseiten erstellt und sie zu Testzwecken in den Serverpfad einbinden will.

Die Klausel `Allow` legt fest, wer Zugriff auf den Server hat. Hier steht normalerweise »from all«. Das bedeutet, dass jeder den Server in Anspruch nehmen darf. In diesem Beispiel ist die Genehmigung auf Rechner eingeschränkt, die der Domäne `willemer.edu` angehören. Diese Einstellung ist typisch für einen Intranetserver. So können Firmeninterna eingestellt werden, die nicht für die breite Öffentlichkeit gedacht sind. Hinter `from` können stehen:

[Zugriffsrechte]L|L Angabe & Bedeutung
all & jeder beliebige Rechner hat Zugriff
.domain.de & Alle Rechner der Domäne `domain.de`
192.168.109.144 & Nur ein spezieller Rechner
192.168. & Alle Rechner, deren IP mit 192.168.109 beginnen

Parallel gibt es noch eine Direktive `Deny from`, die genau das Gegenteil bewirkt, nämlich das Ausschließen von Rechnern oder Netzen.

« [Start des Servers](#) | [Jeder Rechner ein eigener](#) | [Privatadministration per .htaccess](#) »

Privatadministration per .htaccess

Einige der Konfigurationen, die normalerweise zentral in der httpd.conf gemacht werden, können Sie auf der Verzeichnisebene in einer eigenen Datei überschreiben. Allerdings muss das in der zentralen Konfiguration erlaubt sein. Normalerweise wird dort zumindest soviel Freiheit gelassen, dass die Systemsicherheit nicht gefährdet ist.

Interessant ist diese Möglichkeit auch für Kunden von normalen Internet Providern. Denn die Wahrscheinlichkeit ist hoch, dass dort der Apache im Einsatz ist und der Einsatz der .htaccess-Datei erlaubt ist. Damit stehen auch dem normalen Kunden Administrationsmöglichkeiten offen.

In der Datei httpd.conf müssen zunächst ein paar Voraussetzungen geschaffen werden. Mit der Direktive `AccessFileName` wird der Name der Datei festgelegt, die die Konfiguration auf Verzeichnisebene steuert. Normalerweise bleibt es bei dem Standard `.htaccess`. Mit dieser Datei können Direktiven, die normalerweise in der Sektion `Directory` stehen, in die Verzeichnisse direkt ausgelagert werden. Allerdings ist die Voraussetzung, dass die Direktive `AllowOverride` für das Verzeichnis mindestens `FileInfo` hat. Sollen Zugriffskonfigurationen geändert werden, muss auch das Stichwort `AuthConfig` genannt sein.

```
# httpd.conf
AllowOverride FileInfo
```

Die Datei `.htaccess` kann in allen Unterverzeichnissen des Verzeichnisbaums stehen. Unter anderem kann mit der Direktive `ErrorDocument` festgelegt werden, dass bestimmte Fehler auf ein Dokument umgeleitet wird. Besonders interessant ist dabei der Fehler `File Not Found` (404). Es kann eine HTML-Datei angegeben werden, die in solchen Fehlerfällen angesprungen wird. Auf diese Weise können Sie zwar den Fehler nicht verhindern, aber Sie können den Besucher vielleicht auf den korrekten Index setzen, damit er sich wieder zurecht findet.

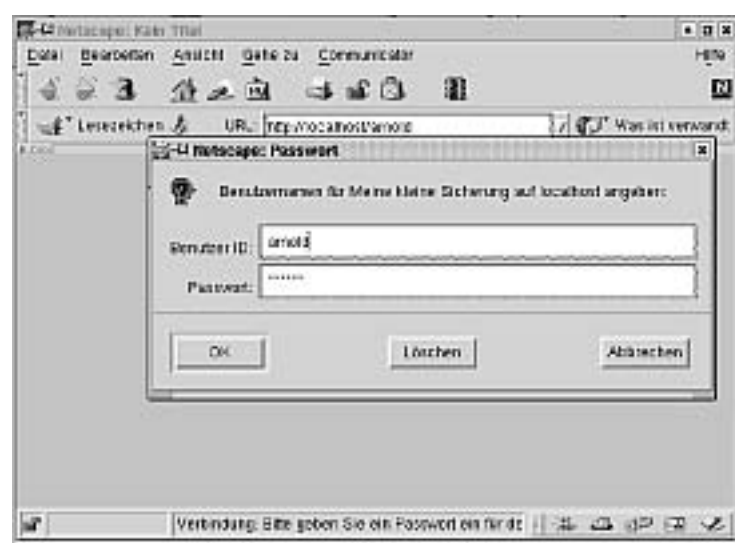
```
# .htaccess
ErrorDocument 404 /err404.html
```

Ein wichtiges Einsatzgebiet der `.htaccess`-Datei ist der Zugriffsschutz für einzelne Unterverzeichnisse. Sie können einstellen, dass bei Aufruf eines Verzeichnisses der Webseite zunächst ein Dialog erscheint, in dem Benutzername und Passwort eingegeben werden müssen. Voraussetzung ist, dass in der `httpd.conf` die Direktive `AllowOverride` mindestens den Eintrag `AuthConfig` aufweist. Dann kann in der `.htaccess` mit mehreren Einträgen die Kennworteingabe vorbereitet werden.

```
AuthType Basic
AuthName "Meine kleine Sicherung"
AuthUserFile htdocs/arnold/.htpasswd
<Limit GET POST>
require valid-users
</Limit>
```

Die Direktiven haben folgende Bedeutung:

- [AuthType] Hier kann Basic oder Digest stehen. Basic verwendet eine unsichere Kodierung des Passwortes, allerdings wird Digest nicht von allen Browsern beherrscht.
- [AuthName] Das ist der Titel der Authorisierung. Er erscheint in der Dialogbox und dient nur der Information des Anwenders, wofür er hier sein Passwort hinterlassen soll.
- [AuthUserFile] beschreibt den Pfadnamen der Passwortdatei (siehe unten).
- [require valid-users] bedeutet, dass nur Anwender zugelassen werden, die in der Passwortdatei unter AuthUserFile aufgeführt sind.



Die Datei .htpasswd enthält für die berechtigten Benutzer je eine Zeile. Jede Zeile beginnt mit dem Benutzernamen. Von einem Doppelpunkt getrennt erscheint dann das kodierte Passwort. Der Name und der Pfad dieser Datei wird durch die Direktive AuthUserFile festgelegt. Als Dateiname verwendet man traditionsgemäß .htpasswd. Der Pfad ist bei einem führenden Schrägstrich der absolute Pfad auf der Maschine. Bei Verwendung eines relativen Pfades geht dieser von dem Pfad aus, der in der zentralen Direktive ServerRoot genannt ist. Wollen Sie einen solchen Schutz in der eigenen Webpräsenz bei einem normalen Provider einbauen, müssen Sie natürlich den Pfad der eigenen Webseite kennen. Freundliche Provider geben diese Information in der FAQ oder auf Anfrage heraus. Wenn Sie freien Zugriff auf den Rechner haben, sollten Sie die .htpasswd an einen Pfad legen, der außerhalb des ServerRoot legen, um ein Ausspähen der Passwörter zu erschweren.

Um die Datei .htpasswd zu erstellen, verwenden Sie am besten das Programm htpasswd. Im folgenden Beispiel wird eine Datei für die Benutzer arnold und andrea erstellt.

```
gaston> htpasswd -bc .htpasswd arnold aBc
Adding password for user arnold
gaston> htpasswd -b .htpasswd andrea PaSw
Adding password for user andrea
gaston> cat .htpasswd
arnold:Htg5PxchrVjLo
andrea:nF9VERTm.Su/Y
gaston>
```

Die Option -c in der ersten Zeile bedeutet create und erzeugt eine neue Passwortdatei. Weitere Benutzer werden nur mit der Option -b angehängt. Der Befehl cat zeigt schließlich, wie die Datei von innen aussieht.

Unterabschnitte

- [Serverantworten](#)

Kommunikation per HTTP

Wie das Zusammenspiel zwischen Browser und Server abläuft, ist in RFC 1945 festgelegt. Ein Webserver tauscht mit dem Browser an sich nur Texte aus. Durch die Möglichkeiten des Browsers bezüglich Hypertext, also des Verlinkens von Informationen und der Einbeziehung von Grafiken und anderer Multimedia wird allerdings eine große Vielfalt an Inhalten dargestellt.

Obwohl mehr Kommandos definiert sind, werden normalerweise hauptsächlich die Befehle GET, POST und HEAD eingesetzt. Bowen, Rich/Coar, Ken: Apache und CGI. Markt und Technik, München, 2000. S. 42. Beim HTTP ist Groß- und Kleinschreibung relevant. GET holt die als Parameter angegebene Datei vom Server und ist damit der Befehl, den ein Browser in fast allen Situationen verwendet. HEAD liest nur den Kopf einer Seite. Dies wird vor allem von Suchmaschinenrobotern verwendet. Der Befehl POST wird für die Übertragung von Daten vom Browser an den Server benötigt, typischerweise in Formularen.

Im folgenden Beispiel wird das Abholen einer Webseite mit Hilfe des Befehls `telnet` simuliert. Zunächst gibt der Client den Befehl GET. Es folgt als Argument die gewünschte Datei. Dabei wird der vollständige Pfad aus Sicht des Servers benötigt. Als dritter Parameter folgt die HTTP-Version. Die Zeilen schließen mit Carriage Return und Line Feed, die durch die Tasten `ctrl-M` und `ctrl-J` bei der Eingabe erreicht werden.

```
gaston> telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /index.htm HTTP/1.0
```

```
HTTP/1.1 200 OK
Date: Mon, 01 Apr 2002 11:03:01 GMT
Server: Apache/1.3.20 (Linux/SuSE) PHP/4.0.6
Last-Modified: Mon, 01 Apr 2002 11:02:40 GMT
ETag: "4b286-7a-3ca83e50"
Accept-Ranges: bytes
Content-Length: 122
Connection: close
Content-Type: text/html
```

```
<HTML>
<BODY>
<H1>Große Überschrift</H1>
```

Schauen Sie sich diesen `Hinweis` an.

```
</BODY>
</HTML>
Connection closed by foreign host.
gaston>
```

Der Server antwortet mit seiner HTTP-Version, der Bestätigung für die Anfrage, diverser Informationen wie Datenumfang und Dateityp. Dann folgt, durch eine Leerzeile getrennt, der Inhalt der Datei.

Sie sehen, dass die Verbindung nach dem Übertragen der Seite getrennt wird. Der Apache ist also ein statusloser Server. Das bedeutet, dass er im Gegensatz zum Newsserver oder Mailserver die Verbindung nach jeder Anfrage abbaut. Das ist zwar etwas einfacher zu implementieren und bringt auch bessere Stabilität und Performance. Der Hauptgrund dafür, dass die Webverbindung statuslos ist, liegt einfach darin, dass diese Sitzungen kein klar definierbares Ende haben. Der Server würde also Verbindungen aufrecht erhalten, deren Client sich längst auf anderen Webseiten bewegt oder einfach seine Sitzung beendet hat. Die Tatsache, dass der Server statuslos ist, hat aber den Nachteil, dass jede Seitenanforderung für den Server neu ist.

In Zeiten der Kommerzialisierung ist es beispielsweise nicht egal, welche Seiten in welcher Reihenfolge aufgerufen werden. Man möchte bei der Programmierung eines Shoppingsystems natürlich gern, dass die Maske mit der Kundenadresse auch dem passenden Warenkorb zugeordnet wird. Das wäre bei Protokollen wie NNTP oder POP3 nicht schwierig, da sich der Client dort an- und abmeldet. Man könnte sich helfen, indem man sich die TCP/IP-Adresse und den Port des Absenders speichert. Leider bleiben diese Werte aber nicht immer erhalten. Der Benutzer könnte eine Wählleitung verwenden, die bei einer gewissen Zeit ohne Datenfluss aus Kostengründen die Verbindung kappt. Sobald dieser Anwender längere Zeit eine Artikelbeschreibung liest, wird die Verbindung abgeschaltet. Wird eine weitere Seite angefordert, baut sich die Verbindung wieder auf und der Anwender bekommt von seinem Provider eine neue IP-Nummer zugewiesen. Damit wäre der Anschluss an die bisherige Sitzung verloren und der bisherige Warenkorb wäre wieder leer. Eine solche Wiedererkennung wird darum oft über den Umweg mit so genannten Cookies erreicht. Cookies sind kleine Dateien, die der Webserver auf dem Arbeitsplatz des Clients speichern und auslesen darf. Damit kann er seinen Kunden eindeutig identifizieren. Da aber einige Server die Cookies verwenden, um das Surfverhalten ihrer Besucher auszuspionieren, haben Cookies einen schlechten Ruf und werden von vielen Besuchern nicht mehr akzeptiert. So identifizieren viele Shoppingprogramme ihre Besucher durch deren Browserausstattung, Grafikauflösung und anderer Informationen.

Serverantworten

Der Server antwortet auf Anfragen mit einer dreistellige Zahl, die über den Erfolg der Anfrage Auskunft gibt. In Tabelle sind die in RFC 1945 definierten Zahlenkombination in einer Übersicht dargestellt.

[Antworten des HTTP-Servers]L|L|L Basis & Zahlencode & Bedeutung

1xx & Informativ
2xx & Erfolgreich
& 200 & OK
& 201 & Created
& 202 & Accepted
& 204 & No Content. Anfrage ok, aber kein neuer Inhalt
3xx & Es sind weitere Aktionen erforderlich
& 300 & Mehrere Auswahlmöglichkeiten
& 301 & Permanent auf anderer URL
& 302 & Temporär auf anderer URL

- & 304 & Inhalt ist unverändert
- 4xx & Fehler des Client
- & 400 & Fehlerhafte Anfrage, etwa ungültige URL
- & 401 & Nicht autorisiert
- & 403 & Verboten
- & 404 & Nicht gefunden
- 5xx & Fehler des Servers
- & 500 & Interner Serverfehler
- & 501 & Nicht implementiert
- & 502 & Falsches Gateway (Proxy-Anwendung)
- & 503 & Server ist überlastet

Wie oben schon erläutert, können Sie beim Auftreten der Fehlermeldungen auf eine eigene Seite verweisen, die nähere Informationen zu dem Problem gibt. Dazu verwenden Sie die Direktive `ErrorDocument`. Darauf folgt zuerst die Fehlernummer, dann das Dokument, das im Falle dieses Fehlers gerufen werden soll.

```
ErrorDocument 404 /err404.html
```

Am wichtigsten dürfte diese Möglichkeit für die Fehlermeldung 404 sein. Beim Umgestalten von Websites werden oft Seiten entfernt, die durchaus noch in den Suchmaschinen vorhanden sind. Mit einer eigenen Fehlerdatei können Sie einen Link auf die Seiten setzen, die vergleichbare Informationen haben. Durch die Möglichkeit, eine `.htaccess`-Datei in jedes Verzeichnis zu legen, kann der Benutzer recht dicht an die Seiten geführt werden, die er nicht gefunden hat.

« [Privatadministration per .htaccess](#) | [Jeder Rechner ein eigener](#) | [Virtuelles Hosting](#) »

Virtuelles Hosting

Das virtuelle Hosting ermöglicht es, mit einem Server mehrere Domänen zu verwalten. Zunächst müssen die Domänen, beispielsweise per DNS, auf denselben Rechner zeigen. Im zweiten Schritt wird in der Konfigurationsdatei `httpd.conf` festgelegt, welche Domänen vom Webserver-Prozess bedient werden. Dazu gibt es die Abschnitte `VirtualHost`. Beispiel:

```
<VirtualHost www.willemer.edu>
ServerName www.willemer.edu
ServerAlias willemer.edu *.willemer.edu
DocumentRoot /www/docs/willemer_edu
</VirtualHost>
```

```
<VirtualHost www.willi.edu>
ServerName www.willi.edu
ServerAlias willi.edu *.willi.edu
DocumentRoot /www/docs/willi_edu
</VirtualHost>
```

Dieser Server wird also die Anfragen an `www.willemer.edu` als auch die an `www.willi.edu` beantworten. Dieses Verfahren wird bei Domainhostern eingesetzt, da diese nicht für jede Kundendomäne einen eigenen Rechner verwenden. Damit wird das Bild von Domänen und Rechnern verdreht. Bisher wurde davon gesprochen, dass mehrere Rechner zu einer Domäne gehören. Hier teilen sich mehrere Domänen einen Rechner. Allerdings ist zu bedenken, dass es für Anwendungen mit IP-basierenden Zertifikaten wie SSL immer noch notwendig ist, eigene IP-Nummern für jeden Hostnamen zu verwenden.

« [Kommunikation per HTTP](#) | [Jeder Rechner ein eigener](#) | [CGI: der Server schlägt](#) »

Unterabschnitte

- [Daten an den Server senden](#)

CGI: der Server schlägt zurück

CGI (Common Gateway Interface) ist eine Schnittstelle zwischen dem Webserver und einer Anwendung. Über diese Schnittstelle kann der Browser Daten an den Webserver übermitteln und der Server liefert dem Browser Daten, die nicht starr sind, sondern durch den Inhalt von Datenbanken oder anderen Applikationen erzeugt werden. Man könnte es so ausdrücken, dass ein normaler Browser zum Frontend einer Applikation wird. Dazu ist es notwendig, dass Webserver und Applikation miteinander sprechen.

Wenn der Browser dynamisch erstellte Daten liefern soll, müssen auf dem Server Programme oder Skripten ablaufen. Der Server muss also auf einen Hinweis im HTML-Quelltext hin diesen Prozess starten. Das Ergebnis des gestarteten Prozesses soll normalerweise wieder auf dem Browser erscheinen. Die einfachste Art, aus HTML einen Prozess auf dem Server zu starten, kann über einen HREF-Link realisiert werden.

```
<A HREF="http://www.server.de/cgi-bin/startmich">Start!</A>
```

Das Verzeichnis cgi-bin liegt typischerweise aus Sicherheitsgründen an anderer Stelle, als die HTML-Dateien für die Webpräsenz. Der Ort des Verzeichnisses cgi-bin wird durch die Konfiguration des Webserver festgelegt. In der Datei httpd.conf findet sich dazu die Zeile:

```
ScriptAlias /cgi-bin/ "/usr/local/httpd/cgi-bin/"
```

Das CGI-Programm startmich ist zu Demonstrationszwecken als einfaches Shellskript realisiert. Der Skript ist leicht zu verstehen, wenn man weiß, dass der Befehl echo alle Parameter auf der Standardausgabe ausgibt.

```
#!/bin/sh
echo "Content-type: text/html"
echo
# Initialisierung beendet: es folgt der Text
echo "<HTML><BODY>"
echo "Es ist gut, Sie hier zu sehen.<br>"
echo "MfG, Arnold Willemer"
echo "</BODY></HTML>"
```

Die Standardausgabe dieses Skripts wird vom Server dem Browser als Ergebnis seines GET-Befehls zugesandt. Wichtig ist die erste Zeile, die den Typ angibt und die darauf folgende Leerzeile, sonst zeigt der Browser nichts an. Danach wird der normale HTML-Code ausgegeben. Als CGI-Sprache eignet sich jede Programmiersprache, die den Standard Input lesen und den Standard Output schreiben kann, also fast jede Sprache. Perl hat eine besondere Bedeutung als CGI-Sprache, weil sie mächtige Befehle zur Verarbeitung von Texten besitzt, und diese im CGI-Bereich besonders hilfreich sind.

Daten an den Server senden

Um das Beispiel zu erweitern, soll die Webseite Daten an den Server senden. Daten gelangen normalerweise über Formulare an den Webserver, die der Anwender in seinem Browser ausfüllt. Ein Formular hat folgende äußere Struktur:

```
<FORM ACTION="/cgi-bin/tuwas" METHOD=GET>
...
</FORM>
```

Der Inhalt dieses Formulars wird von dem Programm `tuwas` verarbeitet. Dieses muss sich auf dem Server in dem für CGIs definierten Verzeichnis befinden. Die Bedeutung der Attribute sind:

- [ACTION=]
Hier wird der Name des Programmes auf dem Server angegeben, das das Formularergebnis bearbeitet. Ausnahme ist die Anweisung `mailto`, die den Inhalt des Formulars an die angegebene Adresse versenden lässt. In diesem Fall sollte die Methode `POST` verwendet werden.
- [METHOD=GET]
Die Daten werden in der Environmentvariable `QUERY_STRING` abgelegt. Das CGI-Programm liest diese Variable aus.
- [METHOD=POST]
Die Daten werden per Standardeingabe an das CGI Programm geliefert. Die Environmentvariable `CONTENT_LENGTH` enthält die Länge der Eingabe.

Das folgende Formular wird nach Drücken auf den Button »Start« das Programm `/cgi-bin/test02` durchstarten.

```
<HTML> <HEAD>
<TITLE>Testseite</TITLE>
</HEAD> <BODY>
```

```
<FORM ACTION="/cgi-bin/test02" METHOD="POST">
<INPUT TYPE="SUBMIT" VALUE="Start">
</FORM>
```

```
</BODY> </HTML>
```

In der Formulardeklaration sehen Sie den Aufruf des CGI-Programmes hinter dem Schlüsselwort `ACTION`. Die Übermittlung der Daten aus der Formularmaske an das Skript erfolgt per Standardeingabe, wenn als Methode so wie oben `POST` verwendet wird. Als Beispiel für die Übermittlung von Daten aus einer Eingabemaske über CGI wird folgendes Formular verwendet:

```
<FORM ACTION="/cgi-bin/test03" METHOD="POST">
<INPUT NAME="Name" VALUE="Name" SIZE=30>
<TEXTAREA Name="Adresse" ROWS=5 COLS=60>
Ihre Adresse
</TEXTAREA>
<INPUT TYPE="RADIO" NAME="Anrede" VALUE="Herr">
<INPUT TYPE="RADIO" NAME="Anrede" VALUE="Frau">
<INPUT TYPE="SUBMIT" VALUE="Start">
</FORM>
```


Die Maske wäre optisch natürlich noch durch eine Tabelle zu strukturieren und erläuternde Texte zu erweitern. Hier soll es aber nur um den Mechanismus gehen und da kann es gar nicht schlicht genug sein. Das Skript wird um das Lesen der Standardeingabe erweitert.

```
#!/bin/sh
echo "Content-type: text/html"
echo
# Initialisierung beendet: es folgt der gesendete Text
read MASKE
echo
```

Der Inhalt des Formulars wird also in die Variable MASKE gelesen und anschließend wiedergegeben. Wir finden dann auf dem Browser die Zeile so wiedergegeben, wie das CGI-Programm sie empfängt.

Name=willemer&Adresse=Ihre+Adresse%0D%0Aort&Anrede=Frau

Sie können anhand der Zeile leicht erkennen, wie der Aufbau der übermittelten Daten aussieht. Die Zuordnung der Elemente hat die Struktur

Feldname=Wert&Feldname=Wert&Feldname=Wert

Die Eingabeelemente werden durch & getrennt. Leerzeichen in Eingabefeldern werden durch ein + ersetzt. Bei den mehrzeiligen Texteingabefeldern (TEXTAREA) werden die Zeilen durch %0D%0A abgeschlossen. Ein kleines Perlskript, das die Auswertung dieser Zeile vornimmt, finden Sie auf S. [cgiperl](#).

Aus den Rohdaten der Formularmaske lassen sich die eingegebenen Werte isolieren. Damit können Sie beispielsweise eine Datenbankabfrage starten. Das Ergebnis können Sie mit HTML-Tags mischen und über die Standardausgabe an den Browser zurückgeben.

Während bei CGI das Programm die Seite als Ausgabe komplett erzeugt, gibt es alternativ auch PHP, das in die Webseite eingebunden wird und nur die zusätzlichen Ausgaben erzeugt, wo der Code steht. Naheliegenderweise sind diese Programme etwas schlanker, da nicht auch alle statischen Seitenbestandteile einzeln programmiert werden müssen. PHP wird nicht als separates Programm gestartet, sondern als Modul. Bei einem intensiv besuchten Server wird dadurch die Belastung geringer. Näheres zu PHP finden Sie auf den folgenden Webseiten:

<http://www.php.net>
<http://www.selfphp.info>

« [Virtuelles Hosting](#) | [Jeder Rechner ein eigener](#) | [Fremdgegangen - Andere Protokolle](#) »

Fremdgegangen - Andere Protokolle

Seit es Netze gibt, gibt es eine Vielzahl von Netzprotokollen. Durch die GNU-Projekte, die vor allem durch Linux noch einmal Auftrieb bekamen, sind unter UNIX die wichtige Clients und Server verfügbar.

Unterabschnitte

- [Samba: UNIX im Windows-Netz](#)
 - [Konfigurationsdatei smb.conf](#)
 - [Starten der Dämonen](#)
 - [Zugriff unter MS Windows](#)
 - [Encrypted Password](#)
 - [Dämon oder inetd](#)
 - [Zugriff auf Windowsnetze](#)
 - [Novell-Zugriffe](#)
 - [Novellclient ncpfs](#)
 - [Der Novellserver Mars](#)
 - [Mac im Netz: netatalk](#)
-

Unterabschnitte

- [Konfigurationsdatei smb.conf](#)
 - [Starten der Dämonen](#)
 - [Zugriff unter MS Windows](#)
 - [Encrypted Password](#)
 - [Dämon oder inetd](#)
 - [Zugriff auf Windowsnetze](#)
-

Samba: UNIX im Windows-Netz

Samba ist eine freie Software, die es UNIX-Rechnern ermöglicht, mit dem Windows Netzprotokoll SMB zu kommunizieren. Samba ist so leistungsfähig, dass Sie es gut verwenden können, um einen Ersatz für einen MS Windows NT Server zu stellen. Es gibt aber auch einen SMB-Client, mit dem Sie Zugriffe auf Windowsressourcen erhalten.

Der Autor von SAMBA, der Australier Andrew Tridgell hatte zunächst seine Software einfach Server und später SMBserver genannt. Als sich herausstellte, dass eine andere Softwarefirma das Namensrecht für SMBserver hatte, suchte er mit grep in einem Wörterbuch nach einen Begriff mit den Buchstaben SMB und stieß auf SAMBA.vgl. die Datei history, die mit SAMBA ausgeliefert wird.

Die Wirkung, die die Veröffentlichung dieser Software hatte, lässt sich heute kaum noch ermessen. Das Network File System (NFS siehe S. nfs), das das Standardnetzwerkdateisystem unter UNIX ist, war von anderen Betriebssystemen nur durch Zusatzsoftware erreichbar. Abgesehen von FTP (siehe S. ftp) gab es also keine Möglichkeit, Dateien zwischen UNIX und MS Windows mit Bordmitteln innerhalb des Netzes auszutauschen. Durch SAMBA war es plötzlich möglich, dass ein Linux Rechner als Ersatz für einen MS Windows NT Server auftrat. Da Sie auf einem Linux-Server im Gegensatz zu NT nicht unbedingt die grafische Oberfläche installieren müssen, ist die Hardwareanforderung erstaunlich gering. Dazu kommt, dass Sie im Gegensatz zum MS Windows NT eine enorme Skalierbarkeit gewinnen. Den Linuxrechner können Sie auch durch leistungsfähige UNIX-Server ersetzen.

Samba arbeitet als TCP/IP-Service. Entsprechend müssen alle SMB-Pakete über das Protokoll TCP/IP transportiert werden. Obwohl in reinen Windowsnetzen oft IPX, das alte Protokoll von Novell, eingesetzt wird, ist das kein so großer Aufwand, da inzwischen fast jeder Windowsrechner auch mit dem Internet verbunden wird und in dem Zusammenhang TCP/IP sowieso installiert wird. IPX können Sie meist ohne Probleme deinstallieren, sofern nicht tatsächlich ein älteres Novellnetz eingesetzt wird. Lediglich MS Windows 3.11 besaß noch nicht von Haus aus TCP/IP. Das können Sie allerdings kostenlos von Microsoft erhalten. Quelle:

<ftp://ftp.microsoft.com/peropsys/windows/public/tcpip/WFWT32.EXE>

Da Samba ein TCP/IP-Dienst ist, müssen in der /etc/services Einträge gemacht werden:

netbios-ssn	139/tcp
netbios-ns	137/tcp

Konfigurationsdatei smb.conf

Die zentrale Konfigurationsdatei heißt `/etc/smb.conf`. Die Datei ist in mehrere Abschnitte eingeteilt und ähnelt einer MS Windows INI-Datei. Die Abschnitte sind mit Titeln in rechteckigen Klammern eingeleitet. Die einzelnen Einstellungen werden durch Zuweisungen gemacht, die der Struktur `Variable=Wert` folgen. Der Wert umfasst den Rest der Zeile. Das Semikolon ist das Kommentarzeichen.

Der wichtigste Abschnitt heißt `global`. Er beschreibt den Server als solchen.

```
[global]
server string = Gaston, der freundliche NT-Ersatz
workgroup = WILLEMER
share modes = yes

; encrypt passwords wird ab NT 4.0 SP3 und win98 gebraucht
encrypt passwords = yes

printing = bsd
printcap name = /etc/printcap

; wichtig, wenn es ein weiteres Interface zum Internet gibt.
interfaces = 192.168.109.144/255.255.255.0
```

Der `server string` ist nur ein Kommentar, der allerdings allen Clients übermittelt wird. Wichtig ist die `workgroup`, die hier originellerweise `WILLEMER` heißen soll. Die Rechner, die zu einer gemeinsamen Arbeitsgruppe gehören, können sich leicht gegenseitig Ressourcen zur Verfügung stellen. Neben den Workgroups gibt es noch das Konzept der Windowsdomänen. Samba kann auch als ein solcher Domänenserver laufen. Nähere Informationen sind in der SAMBA beiliegenden Dokumentation zu finden.

Die beiden Variablen `printing` und `printcap name` ermöglichen den Zugriff auf die in der `/etc/printcap` angegebenen Drucker unter MS Windows. Sie können die Drucker, die mit GhostScript betrieben werden, unter MS Windows als normale PostScript-Drucker ansprechen. Wollen Sie einen speziellen MS Windows"-Druckertreiber verwenden, müssen Sie auf Druckereintrag der Datei `printcap` verweisen, der keinen `if`-Eintrag besitzt.

Im Abschnitt `global` kann eingestellt werden, auf welche Netzinterfaces Samba zugreift. Diese Einstellung ist vor allem bei einem Rechner erforderlich, der auch ein Interface zum Internet besitzt. Ansonsten versucht Samba, auch auf diesem Wege Freunde zu finden. Das wäre natürlich ein eklatantes Sicherheitsrisiko. Hinter dem Schrägstrich steht die Netzmaske (siehe S. subnet). Alternativ kann auch die CIDR-Schreibweise verwendet werden (siehe S. cidr):

```
interfaces = 192.168.109.144/24
```

Im weiteren Verlauf der `smb.conf` werden die Ressourcen beschrieben, die den anderen Rechnern im Netz zur Verfügung gestellt werden. Jede der Ressourcen wird in rechteckigen Klammern genannt.

```
[homes]
comment = Heimatverzeichnis
browseable = no
read only = no
create mode = 0750
```

```
[c-drive]
comment = C-Laufwerk
browsable = yes
public = yes
create mode = 0755
path = /windows/C
writeable = yes
```

Die Ressource `homes` bietet jedem angemeldeten Benutzer sein Heimatverzeichnis an. Meldet sich also der Anwender `arnold` unter MS Windows an und verbindet `homes` mit einem Netzlaufwerk, findet er das Verzeichnis, das laut der `passwd`-Datei unter UNIX sein Heimatverzeichnis ist.

Die Ressource `c-drive` bietet das Verzeichnis `/windows/C` als Netzlaufwerk an. Dieses ist auf meinem Computer tatsächlich eine Windowspartition. Das ist aber keineswegs Voraussetzung. Es kann jedes beliebige Dateisystem verwendet werden.

Die Option `browsable` beschreibt, ob die Ressource angezeigt wird, wenn Sie sie von einem Windowsrechner aus anzeigen lassen. Im Beispiel können Sie also `homes` nicht sehen, wenn Sie sich von einem Windowsrechner die Ressourcen des Servers anzeigen lassen. Allerdings können Sie den Namen `homes` bei einer Verknüpfung direkt angeben. Der Sinn dieser Konfiguration ist es, dass man nicht alle Ressourcen einer breiten Öffentlichkeit darbieten möchte.

Alle Ressourcen müssen natürlich einen Pfad haben, unter dem sie im UNIX Verzeichnisbaum zu erreichen sind. Eingetragen wird dieser Pfad unter der Option `path`. Ausnahmen sind wie gesagt die Heimatverzeichnisse, da sie ihren Pfad über die Datei `/etc/passwd` finden.

Die Option `create mode` stellt ein, welche Berechtigung eine per Samba erzeugte Datei bekommt. Windowsnetze können nicht mit den Berechtigungen umgehen, die UNIX-Dateisysteme kennen. Darum muss vor allem festgelegt werden, welche Rechte die Dateien bei ihrer Erzeugung bekommen. Die Codierung entspricht der, wie sie vom Befehl `chmod` bekannt ist (siehe S. `chmod`).

Wenn die Datei `smb.conf` vollständig konfiguriert wurde, kann Sie durch den Aufruf des Programms `testparm` getestet werden. Das Programm gibt Hinweise auf eventuelle Fehler. Hier der Anfang einer Protokollausgabe des Programms:

```
Load smb config files from /etc/smb.conf
Processing section "[homes]"
Processing section "[c-drive]"
Processing section "[printers]"
Loaded services file OK.
Press enter to see a dump of your service definitions
```

Nach dem Drücken der Return-Taste wird die komplette Konfiguration angezeigt.

Starten der Dämonen

Samba besteht aus zwei Prozessen, `smbd` und `nmbd`. Der Server `smbd` bietet die eigentlichen Dateien- und Druckerdienste an. Der Server `nmbd` ist für die Namensdienste zuständig. Für die ersten Tests können Sie sie einfach direkt nacheinander von der Konsole starten. Mit Hilfe des Befehls `ps` kann in der Prozesstabelle geprüft werden, ob die Prozesse gestartet sind. Ist das nicht der Fall, ist die `smbd.conf` nicht in Ordnung. Nähere Informationen liefern die Dateien `log.smbd` und `log.nmbd`, die sich im Verzeichnis `/var/log/samba` befinden.

Nachdem die Prozesse gestartet sind, wird zum Testen der Konfiguration als nächstes versucht, die Umgebung lokal zu testen. Dazu wird auf derselben Maschine ein SMB-Client gestartet.

```
smbclient -L localhost
```

Nach einer Passwortabfrage werden Informationen über die lokale Samba-Umgebung ausgegeben.

```
Domain=[WILLEMER] OS=[Unix] Server=[Samba 2.0.7]
```

Sharename	Type	Comment
-----	--	----
c-drive		Disk C-Laufwerk
cdrom		Disk Linux CD-ROM
IPC\$ IPC		IPC Service (Gaston, der freund
ascii		Printer cljet5-a4-ascii-mono-300
lp2		Printer cljet5-a4-auto-color-300
lp-mono		Printer cljet5-a4-auto-mono-300
cljet5-a4-raw		Printer cljet5 a4 raw
faxprint		Printer fax

Server	Comment
-----	----

workgroup	Master
-----	----

Zugriff unter MS Windows

Zunächst muss sichergestellt sein, dass der Windowsrechner zur gleichen Workgroup gehört. Als Netzprotokoll muss TCP/IP installiert sein. Die Funktion der Verbindung prüfen Sie zunächst mit dem Befehl `ping`. Der Samba-Server sollte antworten. Wenn dies alles funktioniert, können Sie in der Netzwerkumgebung nachsehen, ob die Workgroup dort erscheint und dann auch der Samba-Server auftaucht. Sie können dem etwas nachhelfen, indem Sie mit der Funktion »Computer suchen« nach dem Server suchen lassen.

Die Netzwerkumgebung ist leider in der Anzeige der verfügbaren Rechner nicht besonders verlässlich. Windowsrechner suchen in gewissen Abständen, welche anderen Rechner sich im Netz befinden. Danach aktualisieren sie ihre Netzwerkumgebung. So kann es sein, dass die angezeigten Rechner längst abgeschaltet sind oder nicht angezeigte Rechner in Wirklichkeit längst verfügbar sind.

Encrypted Password

MS Windows NT 4 ab Service Pack 3 sowie MS Windows 98 und alle neueren Versionen senden

verschlüsselte Passwörter über das Netz. Leider entspricht die Verschlüsselung nicht dem UNIX-Standard, so dass Samba nicht mehr einfach die Datei `/etc/passwd` verwenden kann, wie das mit den unverschlüsselt arbeitenden Windowsversionen noch funktionierte. Zur Speicherung der verschlüsselten Passwörter wird eine eigenständige Datei namens `/etc/smbpasswd` verwendet. Um eine solche Datei zu erzeugen, liefert Samba ein Skript mit, der aus der `passwd`-Datei eine `smbpasswd`-Datei generiert. Der Aufruf lautet:

```
cat /etc/passwd | bash mksmbpasswd.sh >/etc/smbpasswd
```

Danach kann `root` für jeden Benutzer ein Startpasswort mit dem Kommando `smbpasswd` eintragen, indem er beispielsweise für den Benutzer `willemer` eingibt:

```
gaston# smbpasswd willemer
New SMB password:
Retype new SMB password:
Password changed for user willemer.
gaston#
```

Dämon oder `inetd`

Je nach Einsatz gibt es zwei Arten, einen Samba Server zu starten. Wird er nur hin und wieder in Anspruch genommen, reicht es, den Server durch den `inet`-Dämon zu verwalten. Wird der Server intensiv als Firmenserver verwendet, sollten Sie die Dämonen aus den `rc`-Skripten starten.

Die Einträge in der `/etc/inetd.conf` lauten, falls der Dienst auf diesem Wege gestartet werden soll:

```
netbios-ssn stream tcp nowait root /usr/sbin/smbd smbd
netbios-ns dgram udp wait root /usr/sbin/nmbd nmbd
```

Ein `kill -1` auf die PID des `inetd` bringt ihn dazu, seine Konfigurationsdatei noch einmal zu begutachten. Wird nun der entsprechende Dienst angefordert, startet `inetd` die benötigten Samba-Dämonen.

Wird Samba als Server in einer Firma eingesetzt, der regelmäßig benutzt wird, werden die Serverprozesse beim Booten durch eine `rc`-Datei gestartet. In der SuSE-Distribution wird das durch Setzen der Variable `START_SMB` auf »yes« in der `/etc/rc.config` erreicht. Auf anderen Plattformen muss die Datei `smb`, die dem Sambapaket beiliegt, mit symbolischen Links in die richtige Startumgebung gebracht werden (siehe S. `rcskripts`).

Zugriff auf Windowsnetze

Zum SAMBA-Paket gehört auch ein Client. So können UNIX-Rechner auch auf Netzwerklaufwerke von MS Windows oder von SAMBA-Servern zugreifen. Das Programm `smbclient` wurde schon zum Testen des Samba-servers kurz vorgestellt. Mit der Option `-L` Servername zeigt es alle Ressourcen, die der Server anbietet und die als browsable gekennzeichnet sind.

Der Zugriff auf die von `gaston` angebotenen Heimatverzeichnisse würde von einem anderen UNIX-Rechner per `smbclient` ermöglicht

```
smbclient gastonhomes
```

Nach dem Aufruf wird ein Passwort angefordert und Sie befinden sich in einer Umgebung, die stark an den FTP-Client erinnert. Und tatsächlich funktioniert er auch mit den gleichen Kommandos. Etwas irritierend sind die vielen Backslashes. An sich werden vor dem Wort gaston nur zwei davon benötigt. Da aber der Backslash von der Shell interpretiert wird, braucht man hier vier. Allerdings können Sie auch normale Schrägstriche verwenden und dann sieht der Aufruf schon etwas übersichtlicher aus.

```
smbclient //gaston/homes
```

Interessanter als ein FTP-Zugang dürfte aber das Einbinden einer Ressource in den eigenen Verzeichnisbaum sein. Dazu gibt es den Befehl `smbmount`, der einem `mount` mit dem Dateisystemtyp `smbfs` entspricht. Das Einbinden der Ressource HOMES vom Rechner gaston kann also mit einem der beiden Befehle erfolgen:

```
smbmount //gaston/homes /mnt -o username=arnold  
mount -t smbfs -o username=arnold //gaston/homes /mnt
```

Die Verbindung wird mit dem normalen `umount` wieder aufgelöst:

```
umount /mnt
```

« [Fremdgegangen - Andere Protokolle](#) | [Fremdgegangen - Andere Protokolle](#) | [Novell-Zugriffe](#) »

Unterabschnitte

- [Novellclient ncpfs](#)
- [Der Novellserver Mars](#)

Novell-Zugriffe

Die Firma Novell stellt das Produkt Netware her, ein Netzwerkbetriebssystem, das im PC-Bereich lange Zeit vorherrschender Standard war. Trotz der Konkurrenz durch MS-Windows NT Server ist Netware auch heute noch aufgrund seiner Stabilität und seiner Überlegenheit bei großen PC-Netzen verbreitet. Als Novell-Client gibt es für Linux das Paket ncpfs. Es gibt zwei Novellserver: Mars und Iwared. Daneben liefert die Caldera Distribution eine nicht freie Anbindung an Novell mit. Novell ist nicht daran interessiert, sich in die Karten schauen zu lassen. So leiden die Pakete daran, nicht die neuesten Entwicklungen nachvollziehen zu können.

An dieser Stelle wird kurz beschrieben, wie die Pakete installiert werden und wie man eine Verbindung herstellt. Weitere Informationen finden Sie in den den Paketen beigelegten Dokumentationen.

Novellclient ncpfs

Zum Betrieb eines Novellclients muss im Kernel NCP (Novell Core Protocol) als Dateisystem und IPX (Internet Packet eXchange) als Netzwerkoption freigeschaltet sein. Das Paket ncpfs wird automatisch konfiguriert durch folgenden Befehl:

```
ipx_configure -auto_interface=on -auto_primary=on
```

Mit dem Befehl `slist` sehen Sie dann alle Novell-Server im Netz. Wenn das nicht funktioniert, sollten Sie den Kernel kontrollieren.

```
gaston> slist
```

Known Netware File Servers	Network	Node Address

GASTON	C0A86D90	000000000001

```
gaston>
```

Ein Laufwerk von Novell wird mit dem Kommando `ncpmount` eingebunden.

```
ncpmount -S GASTON -U arno1d -P xxxxxx -V SYS /mnt
```

Hinter der Option `-S` steht der Server, nach `-U` folgt der Benutzername. Nach `-P` wird das Passwort angegeben und hinter der Option `-V` wird das Volume, also die Ressource des Servers, angegeben.

Der Novellserver Mars

Der Mars Dämon wird durch den Aufruf von `nwserve` gestartet. Beim Start liest er die Konfigurationsdatei `/etc/nwserve.conf`. In dieser Datei werden die Volumes und weitere Details spezifiziert. Von der Standardanpassung müssen vor allem die exportierten Verzeichnisse (Sektion 1) und der Servername (Sektion 2) angepasst werden.

```
# Syntax:
#      1 VOLUME  DIRECTORY [OPTIONS] [UMASKDIR UMASKFILE]
#      1 SYS     /usr/local/nwe/SYS/   kt   711 600
# Syntax:
#      2 SERVERNAME
#      2 GASTON   # Der Server heißt nun GASTON
```

Hier wird der Server GASTON eingerichtet. Er bietet das Netware-Volume namens SYS an. Die Dateien werden im Verzeichnis `/usr/local/nwe/SYS/` abgelegt. Zuletzt wird die umask festgelegt, getrennt nach Dateien und Verzeichnissen.

« [Samba: UNIX im Windows-Netz](#) | [Fremdgegangen - Andere Protokolle](#) | [Mac im Netz: netatalk](#) »

Mac im Netz: netatalk

Linux kann mit AppleTalk oder genauer EtherTalk kommunizieren und damit für Rechner mit MacOS als Server dienen. Das entsprechende Programmpaket heißt netatalk. Die Grundeinstellung erfolgt in der Datei /etc/atalk/atalk.conf mit der Benennung der Ethernetschnittstelle. Die Anpassungsaufwand ist relativ gering: Es braucht nur die Ethernetkarte eingetragen zu werden. Bei einem Linux-Server steht dort im Allgemeinen nur folgendes:

```
eth0
```

Danach muss der Server `atalkd` und der Server `afpd` gestartet werden. `afpd` liest die Datei `/etc/atalk/AppleVolumes.default`. Darin ist das Verzeichnis `~` eingetragen, also das Heimatverzeichnis. Nachdem der Server `afpd` gestartet wurde, kann jeder Linuxanwender vom Mac auf sein Heimatverzeichnis unter Linux zugreifen.

Um den Macintosh mit dem Linuxrechner zu verbinden, wird unter Classic MacOS das Programm Auswahl gestartet, das sich normalerweise im Apfelmenü befindet. In MacOS X benutzen Sie Verbinden...im Menü Spezial des Finders. Bei Anwahl von AppleShare sollte in der Liste rechts bereits der Name des Linuxrechners auftauchen. Nach der Anwahl erscheint eine Anmeldebox, in der Sie Benutzerkennung und Passwort eingeben können. Das Passwort wird mit der `/etc/passwd` verglichen.

Eine Datei auf dem Macintosh besteht aus zwei Teilen, dem Datenzweig und dem Ressourcenzweig. Um dies auf einem normalen Dateisystem nachzubilden, legt netatalk in jedem Verzeichnis ein besonderes Unterverzeichnis `.AppleDouble` an. In diesem wird der Ressourceanteil der Datei bei einem Zugriff erzeugt.

Firewall und Masquerading

Jedes größere lokale Netzwerk mit einer Verbindung zum Internet ist heutzutage mit einer Firewall abgesichert. Eine Firewall wird oft auf der Basis eines Linuxrechners realisiert. Dieser übernimmt dann normalerweise auch das Masquerading, also die Vermittlung des Zugangs zum Internet, wenn das lokale Netzwerk nicht aus lauter Computern mit internetfähigen IP-Nummern besteht.

Eine wirkungsvolle Firewall kann nur von jemandem konzipiert werden, der TCP/IP komplett verstanden hat. Diese Kurzerläuterung ist darum nicht als Anleitung zum Aufbau einer Firewall fehl zu interpretieren. Aber auch für den einfachen Anwender ist es wichtig zu verstehen, was eine Firewall ist, wie sie arbeitet und wo ihre Grenzen liegen.

Unterabschnitte

- [Funktionsweise einer Firewall](#)
 - [ipchains und iptables](#)
 - [ipfw](#)
 - [Möglichkeiten und Grenzen](#)
 - [Standardpaketfilter](#)
- [Masquerading](#)

Unterabschnitte

- [ipchains und iptables](#)
 - [ipfw](#)
 - [Möglichkeiten und Grenzen](#)
 - [Standardpaketfilter](#)
-

Funktionsweise einer Firewall

Ein Firewall soll einen Einbruch in ein privates Netz verhindern, obwohl das Netz nach außen mit einem öffentlichen Netz verbunden ist. Ist eine TCP/IP-Verbindung einmal hergestellt, wird auch jedes Protokoll transportiert. Ein Router interessiert sich nicht für den Inhalt der Pakete, der er vermittelt.

Vom Prinzip funktioniert eine Firewall so, dass die Verbindung nach außen durchgeschnitten wird und ein Rechner mit zwei Netzwerkschnittstellen dazwischen gesetzt wird. Dieser Computer arbeitet fast wie ein Router. Allerdings schickt er nicht jedes Paket auf die andere Seite, sondern überprüft anhand seiner Regeln, ob das Paket passieren darf, ob es zurückgewiesen werden soll oder einfach nur verworfen wird.

Da jedes Paket die IP Nummer des Senders und des Empfängers enthält und darüber hinaus die Portnummer der beiden, kann der Firewall"-Rechner erkennen, zu welchem Zweck das Paket versandt wurde. Beispielsweise wird eine Anfrage an einen Webserver im Internet als Empfänger eine IP-Nummer haben, die nicht zum Firmennetz gehört und eine Portnummer 80. Der Absender hat eine IP-Nummer aus dem Firmenbereich und als Portnummer eine beliebige Nummer, die nicht zu den well known ports gehört. Bei der Antwort des Servers werden Empfänger und Absender umgetauscht. Dagegen würde ein Versuch, den firmeninternen Intranetwebserver von außen anzugreifen, anders aussehen. Der Empfänger des Pakets hätte eine IP Nummer des internen Firmennetzes und die Portnummer wäre 80. Es ist also an den Paketadressen erkennbar, wer Client und wer Server ist.

Üblicherweise startet man die Firewallregeln damit, jede Kommunikation zu unterbinden. Anschließend werden sukzessive die benötigten und ungefährlichen Kommunikationswege zugelassen. Es muss also der Zugriff auf Webserver über den Port 80 nach außen explizit frei gegeben werden, sonst kann niemand im Internet surfen.

ipchains und iptables

Das Paket ipchains wird mit den Linuxversionen bis Kernelversion 2.2 ausgeliefert. Das ab Version 2.4 verwendete Paket iptables ist weitgehend aufrufkompatibel.

In Regeln wird festgelegt, wie mit einem Paket verfahren werden soll. Dabei gibt es verschiedene Aktionen, die in Abhängigkeit von der Art des Paketes eingeleitet werden können. Die Regeln werden in einem Skript durch den mehrfachen Aufruf von `ipchains` definiert. Dieses Skript wird normalerweise beim Start in den rc-Skripten ausgeführt.

[Aktionen]L|L Aktion & Wirkung
ACCEPT & Paket darf seinen Weg fortsetzen
DENY & Verwerfen und schweigen
REJECT & Ablehnen und Fehlermeldung an Absender
REDIRECT & schickt die Pakete an einen Proxy

Zu Anfang werden alle Regeln gelöscht. Das macht die Option -F (flush).

```
ipchains -F input  
ipchains -F output  
ipchains -F forward
```

Mit der Option -A (add) werden Regeln hinzugefügt. Die folgenden zwei Regeln besagen, dass Pakete von den Schnittstellen eth0 und lo erlaubt sind.

```
ipchains -A input -i lo -j ACCEPT  
ipchains -A input -i eth0 -j ACCEPT
```

Dagegen sind Zugriffe von außen mit den freien Netzwerkadressen verdächtig. An sich werden diese im Internet nicht geroutet, also sollten sie gar nicht auftreten können. Entsprechend kann man von IP-Adressenfälschung ausgehen und solch verdächtige Pakete werden abgelehnt.

```
ipchains -A input -i ppp0 -s 192.168.0.0/16 -j DENY  
ipchains -A input -i ppp0 -s 172.16.0.0/12 -j DENY  
ipchains -A input -i ppp0 -s 10.0.0.0/8 -j DENY
```

Von außen wollen wir alle Zugriffe per telnet (Port 23) ausschließen. Aber SMTP (Port 25) und SSH (Port 22) soll erlaubt sein.

```
ipchains -A input -i ppp0 -p tcp -dport 22 -j ACCEPT  
ipchains -A input -i ppp0 -p tcp -dport 23 -j DENY  
ipchains -A input -i ppp0 -p tcp -dport 25 -j ACCEPT
```

ipfw

ipfw ist eine andere Implementierung einer Firewall. Hier die Kommandos, die Kommunikationswege zu löschen und dann den Zugriff des SSH-Zugriffs über de0.

```
ipfw -f flush  
ipfw add 600 allow tcp from any to any 22 in via de0
```

Letztlich ist das Vorgehen in beiden Fällen identisch, lediglich die Syntax unterscheidet sich. Hier liegt auch derzeit die Problematik der verschiedenen Firewalls. Es gibt keine einheitliche Syntax, obwohl an sich alle Varianten ähnlich arbeiten.

Möglichkeiten und Grenzen

Mit einer Firewall können neben dem Schutz vor Einbrüchen unter anderem auch folgende Ziele erreicht werden:

- Der Internetzugriff kann auf einzelne Rechner beschränkt werden.
- Dienste für das Internet werden auf einzelne Rechner im Netz beschränkt. Dabei kann zwischen den Diensten FTP, WWW, Mail und News unterschieden werden.
- Einzelne Dienste für das Internet können auf einen internen Rechner umgeleitet werden, der beispielsweise als Proxy arbeitet.
- Der gesamte Internetverkehr kann protokolliert werden. Dies betrifft sowohl jeden Angriffsversuch als auch »legale« Verbindungen.
- Aufgrund der Position in der Netztopologie ist der Rechner für das Masquerading prädestiniert.

Trotz der oft gehörten gegenteiligen Annahme ist eine Firewall aber keineswegs in der Lage, einen Vireneinbruch zu verhindern. Ein Virus, das über das Netz eindringt, wird als ausführbare Datei im Anhang einer E-Mail transportiert. Durch das Starten dieser Datei wird meist die Netzwerkkomponente des Betriebssystems verändert. Das ist allerdings bei UNIX unmöglich, da nur der Administrator das könnte. Das ist ein Beispiel dafür, warum ein UNIX-Anwender niemals als arbeitet, wenn es vermeidbar ist. Von dort hängt erzeugt das Virus E-Mails mit sich selbst als Anhang an E-Mailadressen, die es entweder Adressverzeichnissen oder dem Datenstrom, der durch das Socketmodul fließt, entnimmt. Da eine Firewall auf die Ziele und Absender der Pakete, aber nicht auf den Inhalt der Pakete schaut, kann sie einen Virus nicht erkennen.

Des weiteren ist eine Firewall ein Schutz gegen einen Einbruch über das Internet. Es schützt niemals gegen den Angriff aus den eigenen Reihen. Darum ist eine existierende Firewall kein Grund, die Sicherheit der einzelnen Rechner zu vernachlässigen.

Standardpaketfilter

Die Distributionen unter Linux bringen bereits Paketfilter für Rechner mit, die direkt an das Internet angeschlossen werden sollen. Diese Pakete sind zwar nicht individuell auf optimale Sicherheit gestrikt, aber sie sind sicher besser, als die ersten eigenen Versuche. Immerhin ist es ein kostenloser Schutz, auf den Sie nicht leichtfertig verzichten sollten.

Da der Paketfilter auf dem Arbeitsplatzrechner installiert ist, kann er natürlich nicht so sicher sein wie eine separate Firewall. Einen solchen separat stehenden Computer können Sie in seinen Netzverbindungen vollständig beschränken. Das ist mit einem Arbeitsplatzrechner natürlich nicht möglich.

Die Konfiguration einer Firewall sollten Sie nur selbst machen, wenn Sie über eine gute Kenntnis der verschiedenen Protokolle von TCP/IP verfügen. Ferner hilft ein Gespür, an welchen Stellen Sicherheitslöcher entstehen und die Lektüre der aktuellen Sicherheitswarnungen. Unter der folgenden Adresse finden sich regelmäßig die aktuellsten Sicherheitshinweise direkt nach ihrem Bekanntwerden.

<http://www.cert.org/>

Masquerading

Beim Masquerading stellt ein Rechner, der eine gültige IP-Nummer für das Internet besitzt, seinen Zugang zum Internet den anderen Rechnern im Netz zur Verfügung. Ob die IP-Nummer dauerhaft ist oder dynamisch vom Provider zugeteilt wurde, ist belanglos. Der Masqueradingrechner nimmt nun alle Pakete in Richtung Internet in Empfang, steckt sie in die Hülle seiner eigenen Pakete mit gültiger Absendernummer und sendet sie an das Internet. Die zurückkehrenden Pakete packt er wieder aus und sendet sie an den Auftraggeber.

Aufgrund der Nummernknappheit im Internet ist es für eine Firma überhaupt nicht mehr möglich, für alle Rechner eines Firmennetzes eigene internetfähige Nummern zu bekommen. Insofern kommt man am Masquerading heute kaum noch vorbei. Das Masquerading hat auch den Vorteil, dass die Rechner des lokalen Netzes aus dem Internet unerreichbar sind. Da die Anzahl benötigter Adressen gering bleibt, sorgt diese Technik dafür, dass ein Wachstum des Internets möglich ist, auch wenn nicht auf IPv6 umgestiegen werden sollte.

Man kann sich leicht vorstellen, dass das Umsetzen der IP-Nummern am besten an der Stelle geschieht, wo sowieso jedes Paket betrachtet und überprüft wird. Darum wird das Masquerading normalerweise durch die Firewallsoftware durchgeführt. Im folgenden eine Regel unter `ipchains`, die die Pakete des lokalen Netzwerkes 192.168.109.0 weiterleiten.

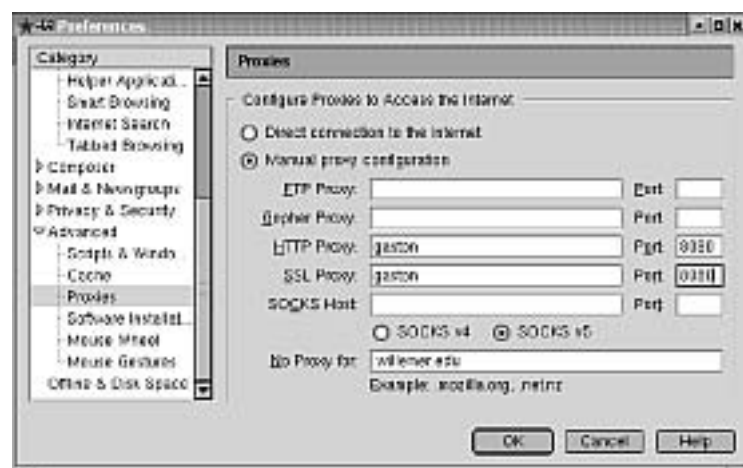
```
ipchains -A forward -i ppp0 -s 192.168.109.0/24  
-d ! 192.168.109.0/24 -j MASQ
```

Mit `ipfwadm` lauten die Befehle: vgl. Barkakati, Naba: LINUX Red Hat 6.0. Franzis', Poing, 2000. S. 578.

```
ipfwadm -F -p deny  
ipfwadm -F -a m -S 192.168.109.0 -D 0.0.0.0/0
```

Proxy

Ein Proxy ist ein Stellvertreter. Die Idee ist, dass die lokalen Rechner ihre Anfragen an den Proxy stellen und dieser die Anfrage ans Internet weiterleitet. Am häufigsten wird ein Proxy für HTTP, als das World Wide Web eingesetzt. Da die Rechner im Netz ihre Anfragen an den Proxy und nicht direkt ans Internet stellen, ist weder Routing noch eine Anbindung an das DNS erforderlich, um den Arbeitsplätzen zu ermöglichen, im Internet zu surfen. Alle gängigen Browser lassen sich leicht auf den Proxy-Betrieb umstellen.



In der Abbildung wird ein Browser auf Proxy-Betrieb umgestellt. Dabei werden alle Seitenanforderungen, die per HTTP oder SSL aufgerufen werden, zunächst an den Computer gaston gesendet. Der Zielport ist auf gaston 8080 statt dem üblichen Port 80 für HTTP. Der Port 8080 wird auf gaston von dem Proxy bedient. Er empfängt dort die Anforderungen und sendet diese weiter ins Internet. Die Antworten, die aus dem Internet kommen, werden wiederum an den ursprünglichen Client zurückgesandt.

Der Proxy bietet mehrere Vorteile:

- In einem LAN braucht nur der Proxy-Computer den Zugang zum Internet zu haben. Das Routing und die DNS-Konfiguration muss nicht für das Internet angepasst werden.
- Ein Angreifer aus dem Internet kann nur den Proxy erreichen. Sofern er hier nicht einbrechen kann, ist das restliche Netz sicher.
- Der Proxy übernimmt normalerweise auch das Caching. Werden also immer wieder die gleichen Seiten aus dem Web aufgerufen, wird der Zugriff beschleunigt.
- Bestimmte Webseiten können ausgefiltert werden
- Man erhält ein Protokoll, wer wann welche Webseiten aufgerufen hat.

Apache als Proxy

Der HTTP-Server Apache kann als Proxy eingesetzt werden. Wenn also auf dem Rechner, der die Internetverbindung aufbaut, bereits ein Apache läuft, ist es sinnvoll, diesen als Proxy einzurichten. Der Proxy ist ein eigenständiges Modul für Apache, das natürlich installiert sein muss. Die Konfiguration erfolgt, wie bei Apache üblich, in der Datei /etc/httpd/httpd.conf.

```
<IfModule mod_proxy.c>  
    ProxyRequests On
```

```
<Directory proxy:*>
    Order deny,allow
    Deny from all
    Allow from .willemer.edu
</Directory>
ProxyVia On
</IfModule>
```

Die Einträge sind nicht weiter kompliziert. ProxyRequests schaltet die Anfragen an den Proxy frei. Dann erfolgt im Directory proxy die Sicherheitseinstellung. Nur Rechner aus der Domäne willemer.edu dürfen diesen Proxy benutzen.

squid

Für Umgebungen mit einem umfangreicheren Verkehr ins Internet empfiehlt sich ein spezialisierter Proxy. Ein renommierter Proxy nennt sich squid. Konfiguriert wird er durch die Datei /etc/squid.conf. Es liegt dem Paket eine Beispieldatei bei, die Sie am besten weitgehend unverändert lassen.

Beim Start prüft squid die DNS-Umgebung für das Internet. Hat er beispielsweise in einer Umgebung mit einer Wählleitung keinen Zugriff darauf, sollten Sie beim Start die Leitung frei schalten. Vor dem ersten Start sollten Sie squid einmal mit der Option -z starten. Dann erzeugt er alle Verzeichnisse, die er zum Caching der Webseiten benötigt. In der Anpassung sind die fett gesetzten Zeilen in der squid.conf hinzu gekommen:

```
acl acldom srcdomain    .willemer.edu
acl aclnetz src 192.168.109.0/255.255.255.0
#
http_access allow manager localhost
http_access deny manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
# INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS FROM YOUR CLIENTS
http_access allow acldom
http_access allow aclnetz
# And finally deny all other access to this proxy
http_access allow localhost
http_access deny all
```

Im Beispiel werden die acl-Variablen acldom und aclnetz definiert. Das erste wird über die Domäne, das zweite über die IP-Nummer. Diese Variablen werden weiter unten für den http_access verwendet, so dass beiden der Zugriff auf den Proxy ermöglicht wird. Ansonsten würde nur localhost einen Zugriff bekommen. An sich ist das natürlich redundant. Der Zugriff ist einmal für die IP-Nummern des Netzes 192.168.109.0 und einmal für die Domäne willemer.edu freigeschaltet worden.

Neben ihrer Funktion als Sicherung gegen Einbrüche aus dem Internet und der Weiterleitung von Internetanfragen haben Proxies auch eine Caching-Funktion. Sie speichern also alle angeforderten Seiten für einige Zeit, und holen sie hervor, wenn die Seite ein zweites Mal geladen wird. Da beispielsweise Suchmaschinen und bestimmte Fachinformationen immer wieder aufgerufen werden, ergibt sich eine gewisse Beschleunigung.

Für den Proxy squid wird ein zusätzliches Produkt mit dem Namen squidguard angeboten. Dieses Programm dient als Filter. Damit lassen sich bestimmte Rechner freischalten oder sperren und bestimmte Begriffe aus den URL-Namen ausfiltern. Sie können darüberhinaus Seiten für bestimmte

Tageszeiten ausschließen. Die genaue Konfiguration können Sie den mitgelieferten Dokumentationen entnehmen.

Ein Proxy protokolliert genau, wer welche URL aufgerufen hat. Es lassen sich also Statistiken erstellen, wer wann welche Seiten betrachtet hat. Insofern kann es sein, dass die Firma einen Proxy nicht nur als Schutz gegen das Internet verwendet, sondern auch zur Kontrolle seiner Angestellten.

Unterabschnitte

- [Apache als Proxy](#)
- [squid](#)

« [Masquerading](#) | **Netzwerk** | [Das X-Window System](#) »

Das X-Window System

- [Das X-Window System](#)
 - [Grafische Oberfläche unter UNIX](#)
 - [X Window starten](#)
 - [X Window benutzen](#)
 - [Konfigurieren](#)
 - [Desktops](#)
 - [Das X Window System im Netz](#)
 - [Anwendungssoftware für UNIX](#)
-

Das X-Window System

Die grafische Oberfläche zu UNIX sah immer etwas karg gegenüber der bunten Welt der kleinen Computer aus. Dies ist erst in letzter Zeit ein wenig anders geworden. Dafür sind die inneren Werte weit überlegen.

- Grafische Oberfläche unter UNIX
 - X, Fenstermanager, Widget Sets und Desktop
 - Der X-Server
 - Der Fenstermanager
 - Der X-Client und seine Bibliotheken
- X Window starten
 - Nacktstart mit xinit
 - Regulärer Start von X: startx
 - Grafisches Einloggen: Display Manager xdm
 - Eine kleine Beispielsitzung mit xdm
- X Window benutzen
 - Bedienungselemente des Athena Widget Set
 - Der Aufruf von X-Programmen
 - Cut and Paste
 - Terminalfenster xterm
 - Weitere praktische Helfer
- Konfigurieren
 - Farbbeschreibung
 - Schriften
 - Bitmaps
 - Ressourcen
 - Konfiguration des Fenstermanagers
 - Fokus und Z-Anordnung
- Desktops
 - CDE
 - KDE
 - GNOME
 - Der Wettstreit der freien Desktops
 - MacOS X
- Das X Window System im Netz

- X-Programme über Netz starten
 - X-Server Software in Betrieb nehmen
 - Grafisches Einloggen übers Netz
 - Thin Client
 - Anwendungssoftware für UNIX
-

Grafische Oberfläche unter UNIX

UNIX hatte recht früh eine grafische Oberfläche. Das X Window System wurde 1984 von Robert W. Scheifler und James Gettys am Massachusetts Institute of Technology (MIT) entwickelt.vgl. Mikes, Steven: X Window System Program Design and Development. Addison-Wesley, Reading, 1992. p. 5 Dennoch wurden die grafischen Oberflächen unter UNIX immer etwas stiefmütterlich behandelt. Der Grund ist, dass UNIX ursprünglich nur auf teuren Maschinen lief, die entweder ihren Ort als Server im Unternehmen fanden, und dort keine Grafik brauchten. Oder sie wurden als Workstation für grafische Anwendungen verwendet. Dann waren es aber sehr spezielle Anwendungen wie CAD oder Simulationen, die weniger auf den Komfort des Benutzers ausgerichtet waren, sondern für Spezialisten entwickelt wurden, die eine leistungsfähige Maschine benötigten.

Der Arbeitsplatz einer Sekretärin wurde erst dann mit einem Computer versehen, als die verhältnismäßig günstigen PCs verfügbar waren. Und diese Geräte wurden auch erst Anfang der 90er Jahre mit der grafischen Oberfläche MS Windows ausgestattet. Zwar hatte der Apple Macintosh bereits 1984 über Grafik und Maus verfügt, aber auf den Schreibtischen der normalen Sekretärin war er eher die Ausnahme.

Erst nachdem die Fenster im Büro die Regel waren, begann man die grafische Oberfläche von UNIX unter ergonomischen Gesichtspunkten zu betrachten. Bis dahin wurde das X Window System entweder von besagten Spezialisten bedient oder von Programmierern, um auf dieser Plattform Software zu entwickeln. Beiden konnte man zumuten, dass die Oberfläche nicht unbedingt immer intuitiv zu bedienen war.

Die wichtigsten Merkmale des X Window Systems sollen hier im Überblick genannt werden.

- X ist netzwerkfähig. Tastatur, Maus und Grafikschild müssen an der Maschine gar nicht vorhanden sein. Es reicht aus, irgendwo im Netzwerk einen Computer zu haben, die mit Maus und Grafik umgehen kann.
- X arbeitet mit austauschbaren Windowmanagern und Desktops.
- Jeder Anwender kann seinen Desktop völlig unabhängig von anderen Benutzern konfigurieren.
- Man kann eine beliebig große Auflösung als virtuellen Bildschirm verwenden. Der reale Bildschirm folgt der Mausbewegung wie ein Sichtfenster darüber hinweg
- Man kann zwischen mehreren Arbeitsbildschirmen umschalten.
- Farben, Zeichensätze und Beschriftungen können für die meisten Programme vom Anwender verändert werden.

Zum Namen des X Window System lässt sich anmerken, dass Sie besser kein s an das Wort Window hängen. Stellen Sie zum Test in einer Newsgroup eine knifflige Frage zum Thema X und schreiben Sie dabei einmal X Windows. Sie werden zu 95% Antworten von Leuten bekommen, die der Ansicht sind, dass es für Sie wichtiger ist zu lernen, wie man X Window richtig schreibt, als dass Ihr Problem gelöst wird. Es soll inzwischen sogar Leute geben, die den Namen im Gespräch absichtlich falsch aussprechen und das s noch besonders betonen, um heraus zu finden, ob ihr Gegenüber ein Besserwisser ist.

- [X, Fenstermanager, Widget Sets und Desktop](#)
- [Der X-Server](#)
- [Der Fenstermanager](#)
- [Der X-Client und seine Bibliotheken](#)
 - [Xlib und X Toolkit Intrinsics](#)
 - [Widget Sets](#)

« [Das X-Window System](#) | **[Das X-Window System](#)** | [X, Fenstermanager, Widget Sets](#) »

X, Fenstermanager, Widget Sets und Desktop

Die grafische Oberfläche unter UNIX ist sehr leistungsfähig, und darum auch sehr vielschichtig.

Das X Window System besteht aus dem X-Server und dem X-Client. Der X-Server ist der Dienstleister, der dem System einen grafischen Bildschirm, eine Maus und eine Tastatur zur Verfügung stellt. Das Programm, das für eine grafische Oberfläche geschrieben wurde, nimmt die Dienste des X-Servers in Anspruch und ist damit der X-Client. Ein X-Client ist also immer ein Programm, das für X geschrieben wurde. Damit der X-Client seine Anforderungen an den X-Server stellen kann, gibt es das X-Protokoll. Dieses kann lokal verwendet werden, wenn X-Client und X-Server auf dem gleichen Rechner laufen. Solche Computer nennt man Workstations. Das X-Protokoll kann aber auch über ein Netzwerk übermittelt werden. Dann befinden sich X-Server und X-Client auf unterschiedlichen Rechnern. Als X-Terminal bezeichnet man ein Gerät, das ausschließlich als X-Server verwendet werden kann.

Ein Widget Set ist Satz von Widgets. Widgets sind Programmiermodule, aus denen die meisten Fenster und Dialogboxen unter X zusammengesetzt sind. Die sichtbaren Widgets sind die Kontrollelemente wie Buttons, Eingabefelder oder Schieberegler. Die weniger anschaulichen Widgets sind die Containerwidgets. Sie dienen der Anordnung der Kontrollelemente. Dabei gibt es solche, die alle Elemente nebeneinander anordnen, andere, die ein Gitter vorgeben und andere, mit denen komplexeste Anordnungen zu konstruieren sind. Diese Container sind charakteristisch für X, weil sie dafür sorgen, dass die Elemente bei Größenveränderung des Fensters nicht starr festliegen, sondern scheinbar eine ideale Aufteilung des vorhandenen Raums suchen. Aufgrund der Container sind unter X auch Dialogboxen normalerweise in ihrer Größe vom Anwender veränderbar. Diese Widgets sind in einem Widget Set zusammen gefasst. Der Widget Set bestimmt das Design und auch das Verhalten der einzelnen Widgets. Die Wahl des Widget Sets wird für jedes Programm vom Programmierer getroffen. Es ist völlig normal, wenn unter X mehrere Programme parallel laufen, die mit unterschiedlichen Widget Sets erzeugt wurden. Das wohl bekannteste Widget Set ist Motif.

Der Fenstermanager ist der Bestandteil des X Window System, der den Rahmen von Fenstern liefert. Darüber hinaus ist er zuständig für das Verschieben oder Auszuwählen von Fenstern. Mit dem Widget Set zusammen bestimmt der Fenstermanager in entscheidender Weise das Look and Feel. So gehört zu dem Motif Widget Set auch der Motif Window Manager (mwm). In der Wahl des Fenstermanagers ist X flexibel. Viele Liebhaber anderer grafischer Umgebungen haben »ihre« Umgebung mit speziellen Fenstermanagern oder nur durch Umkonfigurieren bestehender Fenstermanager auf X übernommen. So gibt es Umgebungen, die sich verhalten wie der Presentation Manager von OS/2, wie der Amiga oder wie NextSTEP.

Anfänglich dienten die Fenstermanager nur als Startrampen für X"-Programme. Inzwischen wurden immer mehr Elemente hinzugefügt, um ein einheitliches Erscheinungsbild zu erreichen. Auch die Kommunikation zwischen den Programmen wurde vorangetrieben.

Der X-Server

Normalerweise ist der Server der Rechner im Hintergrund und der Client befindet sich direkt vor der Nase. Unter X ist das umgekehrt. Ein X-Server bietet einen grafikfähigen Bildschirm, eine Tastatur und eine Maus an. Ein X-Client als Nutzer dieser Ressourcen ist ein Programm, das eine grafische Darstellung wünscht.

Als X-Server wurden vor einigen Jahren noch echte X-Terminals eingesetzt, die nichts weiter konnten, außer Grafik darstellen und Netzverbindungen anzulegen. Inzwischen lohnt sich der Bau solcher Geräte nicht mehr. X-Server sind heutzutage typischerweise Programme. Diese laufen auf Macintosh oder MS Windows. Oder man nutzt den eingebauten X-Server eines UNIX-Rechners.

Da die Anforderungen des X-Clients auf einer recht abstrakten Ebene stattfinden, werden die Transportpakete relativ klein gehalten. Es werden also nicht die Speicherabzüge der Bildschirme über das Netz versandt, sondern beispielsweise die Koordinaten und der Farbcode eines Rechtecks.

Das Programm X-Server heißt auf einer UNIX-Maschine einfach X und steuert Bildschirm, Maus und Tastatur an. Der Server verbirgt die Hardware vor den Anfragen der X-Clients.

[« X, Fenstermanager, Widget Sets | Grafische Oberfläche unter UNIX | Der Fenstermanager »](#)

Der Fenstermanager

Der Fenstermanager (engl. Window Manager) steuert das Verhalten des X Window System. Er ist für das Zeichnen der Rahmen und das Positionieren der Fenster zuständig. Das Positionieren ist unter X aufwändig. Wird ein Fenster mit Kontrollelementen erzeugt, werden die Platzanforderungen zwischen Rahmen und Elementen ausgeglichen. Kontrollelemente sind unter X als Widgets realisiert, die sich fast wie Fenster verhalten. Ihre Position innerhalb eines Fensters werden durch Containerwidgets realisiert. Diese Container nehmen Widgets auf und ordnen sie an. Dabei unterscheiden sich die verschiedenen Containerwidgets in der Strategie, wie die Widgets angeordnet werden. Um die Anordnung der Widgets innerhalb des Fensters zu regeln, berechnen die Kontrollelemente ihren Platzbedarf und leiten ihn an die Containerwidgets weiter. Diese leiten es wiederum an die Container, in denen sie selbst liegen, bis die Größe des benötigten Platzes beim Rahmen angekommen ist. Diese Größe wird dem Fenstermanager mitgeteilt. Soll das Fenster neu erstellt werden und reicht der Platz auf dem Bildschirm, wird er den Wunsch bestätigen. Ansonsten wird er den äußeren Containerwidgets den verfügbaren Platz nennen. Diese werden ja nach ihrer Anordnungsstrategie den Platz auf die Widgets verteilen und ihnen die neue Größe mitteilen. Zuletzt werden sich die Kontrollelemente in dem verfügbaren Raum darstellen.

Der Fenstermanager kontrolliert die Reaktionen auf die Standardereignisse. So wird im Fenstermanager konfiguriert, welche Menüs beim Klicken auf den Hintergrund erscheinen. Die Strategie des Fokuswechsel (siehe S. fokus) und die Tastaturanpassung werden ebenfalls im Fenstermanager konfiguriert.

Es gibt diverse Fenstermanager. Der erste, der auch in der Basisausstattung von X beilag, heißt uwm. Heute wirkt uwm etwas archaisch. Die Fenster bekommen weder Titelleiste noch Rahmen. Sie können ein Fenster verschieben, indem Sie in die Fenstermitte klicken und dann ziehen. Über Kontextmenüs können Sie Fenster in ihrer Größe verändern. Ein etwas komfortablerer Fenstermanager ist twm. Im professionellen Bereich war der Motif Window Manager (mwm), der mit dem Motif Widget Set kam, lange Standard und ist heute noch die Basis von CDE, dem kommerziellen UNIX Desktop. Linux brachte neben vielen anderen den Fenstermanager fvwm, den es wiederum in mehreren Versionen und Varianten gibt. KDE hat einen eigenen, speziell auf seine Bedürfnisse angepassten Fenstermanager. Für GNOME wurde der Fenstermanager Sawfish entwickelt.

Unterabschnitte

- [Xlib und X Toolkit Intrinsics](#)
- [Widget Sets](#)

Der X-Client und seine Bibliotheken

Ein X-Client ist jedes Programm, das für X geschrieben wurde. Aus der Sicht des Programmierers unterscheidet sich die Programmierung einer Anwendung für X nicht grundlegend von der für MS Windows oder für Macintosh. Die mögliche Netzverteilung, die Konfigurierbarkeit und vieles mehr übernehmen die Bibliotheken, die beim Erstellen des Programmes eingebunden werden.

Xlib und X Toolkit Intrinsics

Einer Anwendung, also einem X-Client, ist die Kommunikation mit dem X-Server, die ja auch über das Netz gehen kann, vollkommen verborgen. Diese Leistung erbringt die Bibliothek Xlib. Die Xlib enthält alle Bestandteile, die zu einer normalen Grafischicht gehören. Alle Grafikausgaben werden über das X-Protokoll an den X-Server versandt. Die Xlib verfügt darüber hinaus über einige Objekte. So kennt die Xlib beispielsweise Fenster und Mauszeiger. Allerdings sind die Fenster der Xlib nicht viel mehr als Rechtecke, die die Ereignisse empfangen, die in ihren Grenzen ausgelöst werden. Es ist möglich, Applikationen zu schreiben, die allein auf der Xlib basieren. Ein Beispiel dafür ist `xcalc`, der Taschenrechner, der bei jeder X-Auslieferung zum Standardumfang gehört. Allerdings ist eine X-Anwendung, die nur auf der Xlib basiert, eher untypisch, da sie aufwändig zu programmieren ist.



Die Toolbox Intrinsics ist eine Bibliothek, die auf der Xlib aufsetzt und die Verwaltung von Widgets übernimmt. Sie stellt selbst kein Widget-Set zur Verfügung, sondern nur das Grundgerüst und die Grundfunktionen zur Erstellung eines Sets. Daneben verwaltet sie die Eigenschaften der Widgets und stellt dem Anwendungsprogrammierer die Funktionen zum Erzeugen, Darstellen und Entfernen von Widgets zur Verfügung.

Eine Aufgabe der Toolbox liegt in der Verwaltung der Eigenschaften von Widgets. Diese so genannten Ressourcen betreffen Farbe, Ausdehnung, Beschriftung und Ereignisauslösung. Sie können durch frei editierbare Dateien, die so genannten Ressourcendateien, verändert werden. Die Verwaltung dieser

Einstellungen wird durch die Intrinsics "übernommen.

Widget Sets

Ein Widget Set ist sozusagen der Satz von Bausteinen, aus denen ein X-Programm zusammengesetzt wird. Sichtbar wird ein Widget Set vor allem an den Kontrollelementen. Das Aussehen der Kontrollelemente hat einen erheblichen Einfluss auf den Gesamteindruck der Applikation. Das Widget Set und die darunter liegenden Bibliotheken liegen normalerweise als dynamische Libraries auf dem Rechner.

Motif war einige Zeit der Standard für kommerzielle X-Anwendungen. Man fand Motif auf jedem relevanten UNIX System. Lediglich unter Linux ist es selten vorhanden, da Lizenzgebühren anfallen. Dennoch kann man Motif-Programme auch unter Linux laufen lassen, wenn der Programmierer die Library statisch zum Programm bindet. Dadurch werden die Programme natürlich sehr groß und unhandlich.

Das Athena Widget Set ist eines der ältesten Widget Sets, aber es hat den Vorteil, auf jedem X-Window-System kostenlos verfügbar zu sein. Aus diesem Grund sind vor allem die grundlegenden Programme oft mit diesem Widget Set erstellt. Leider ist das Widget Set sehr einfach und lässt viele Kontrollelemente vermissen. Dazu kommt ein für den heutigen Geschmack sehr tristes Aussehen. Unter Linux gibt es eine Variante des Athena Widget Set, die mit dem heute üblichen 3D-Effekt verschönert wurde. Alle Programme, die den Athena Widget Set verwenden, erhalten dieses Aussehen, wenn dieser Set installiert ist.



Als das Athena Widget Set in die Jahre gekommen war, schlossen sich einige UNIX-Hersteller zusammen, um einen kommerziellen Standard für ein Widget Set zu schaffen. Das Ergebnis war Motif. Motif ist nicht nur wesentlich eleganter im Aussehen, es brachte auch neue Kontrollelemente mit, die man beim Athena Widget Set schmerzlich vermisste. Vor allem aber hatte Motif eine andere Benutzerphilosophie. Mit Motif wurde die Bedienung von X an die des Macintosh und damit die von MS Windows angepasst. Motif war allerdings lizenzpflichtig, was dazu führte, dass es unter Linux nur für diejenigen eine Rolle spielte, die kommerzielle Software entwickelten und eine preiswerte Entwicklungsumgebung brauchten. Ein Beispiel für das Aussehen einer typischen Motif-Anwendung ist das HP-UX Administrationstool sam, das Sie auf Seite sampic finden.

Die Widget Sets sind also dafür verantwortlich, dass manche Anwendungen ganz andere Buttons

besitzen als andere. Auch die Tatsache, dass sich bei manchen Programmen die Rollbalken völlig anders bedienen lassen, geht auf unterschiedliche Widgets zurück.

« [Der Fenstermanager](#) | [Grafische Oberfläche unter UNIX](#) | [X Window starten](#) »

X Window starten

X kann auf unterschiedliche Weise gestartet werden. Sie können aus einer normalen Anmeldung heraus den Grafikschirm durch den Aufruf von `startx` starten oder beim Booten bereits in einen grafischen Login gelangen, um sich von dort direkt in seine Sitzung einzuloggen.

Unterabschnitte

- [Nacktstart mit xinit](#)
 - [Regulärer Start von X: startx](#)
 - [Grafisches Einloggen: Display Manager xdm](#)
 - [Eine kleine Beispielsitzung mit xdm](#)
 - [Varianten des xdm](#)
-

Nacktstart mit xinit

In diesem Abschnitt soll ein wenig experimentiert werden, um zu zeigen, wie X startet. Starten Sie dazu von der Kommandozeile den X-Server durch den direkten Aufruf von `xinit`. `xinit` ist ein Programm, das normalerweise durch `startx` gestartet wird. Bevor Sie dies tun, müssen Sie aber die Datei `.xinitrc` im Heimatverzeichnis umbenennen. Im Anschluss an das Experiment sollten Sie nicht vergessen, die Datei zurück zu benennen.

```
cd
mv .xinitrc .xinitrc.orig
xinit
..
mv .xinitrc.orig xinitrc
```

Nach dem Aufruf von `xinit` wird der Bildschirm grau. Es erscheint ein fettes, diagonales Kreuz als Cursor und in der linken oberen Ecke befindet sich ein helles Rechteck, in dem sich offensichtlich ein Shellprompt befindet. Wenn Sie mit dem Mauszeiger in das helle Feld fahren, wird der Cursor offensichtlich aktiv. Es wird Sie vielleicht überraschen, aber dieses helle Feld ist ein Fenster. Es fehlt ihm nur der Fensterrahmen, den normalerweise der Fenstermanager zur Verfügung stellt. Den Fenstermanager können Sie an dieser Stelle von Hand aufrufen. Je nachdem, welcher Fenstermanager auf ihrem System installiert ist, starten Sie `mwm` (Motif, CDE) oder `kwm` (KDE), `sawfish` (GNOME), `fvwm` (Linux), `twm` oder `uwm` (alter MIT Fenstermanager). Danach sollte sich das Bild verändern. Vermutlich erscheinen Fensterrahmen und nun ist es auch möglich, das helle Fenster zu verschieben. Sofern Sie kein `&` hinter den Aufruf des Fenstermanagers gesetzt haben, wird dieser durch `ctrl-C` oder die `Del`-Taste wieder verschwinden. Sobald Sie sich aus dem Fenster ausloggen, wird auch `xinit` und damit der X-Server wieder enden.

[« X Window starten](#) | [X Window starten](#) | [Regulärer Start von X: »](#)

Regulärer Start von X: startx

Wenn UNIX in den Runlevel 2 bootet, finden Sie eine textuelle Umgebung vor. Um X aus der Shell zu starten, verwenden Sie den Befehl `startx`. Dieser startet den Prozess `xinit`, der die Datei `.xinitrc` im Heimatverzeichnis des Benutzers ausführt und anschließend endet. Ist keine Datei `.xinitrc` im Heimatverzeichnis des angemeldeten Benutzers vorhanden, wird die Systemdatei `/usr/X11/lib/X11/xinit/xinitrc` verwendet. In dieser Datei stehen die Startaufrufe aller Programme, die mit dem X auf dem Bildschirm erscheinen sollen. Alle diese Programme werden in den Hintergrund mit einem `&` gestellt. An letzter Stelle muss aber ein Programm ohne `&` gestartet werden und das ist normalerweise der Fenstermanager, dessen Namen meist auf »wm« endet. Er wird mit dem Kommando `exec` aufgerufen. Das führt dazu, dass er an die Stelle des Prozesses `xinit` tritt. Dadurch, dass er in der Datei `xinitrc` an die Stelle des `xinit`-Prozesses tritt, endet die X-Sitzung, sobald der Fenstermanager beendet wird.

Damit ist `.xinitrc` die wichtigste Konfigurationsdatei. Sie gestaltet den Initialdesktop und bestimmt den Fenstermanager.

[« Nacktstart mit xinit | X Window starten | Grafisches Einloggen: Display Manager »](#)

Grafisches Einloggen: Display Manager xdm

Eine Workstation wird heutzutage selten in den Runlevel 2 booten, sondern gleich die grafische Oberfläche starten und ein grafisches Login anbieten. Der Display Manager xdm wird durch den Übergang in den Runlevel 3 oder wie bei Linux im Runlevel 5 (siehe S. runlevel) gestartet. Entsprechend kann der Start von xdm einfach durch einen Eintrag in der /etc/inittab erreicht werden. vgl. Eßer, Hans-Georg: KDE Der neue Desktop für Linux. Sybex, Düsseldorf, 1999. S. 95

```
x:5:respawn:/usr/bin/x11/xdm -nodaemon
```

Oft wird xdm aus einem Initskript gestartet. Der Link befindet sich dann im Verzeichnis /etc/rc.d/rc3.d, wenn xdm im Runlevel 3 gestartet wird.

Zwei Dateien bestimmen den Bildschirm des Anwenders nach dem Einloggen per xdm: .Xsession und .Xdefaults. Die Datei .Xsession wird wie die .xinitrc verwandt. Es werden Applikationen gestartet. Ist die .Xsession abgearbeitet, endet auch die Sitzung und der Anmeldebildschirm erscheint. Darum werden die Programmaufrufe so sortiert, dass der letzte das Programm startet, das läuft, solange die Sitzung läuft. Meist ist das der Fenstermanager. Dieses Programm wird mit dem Befehl exec gestartet. Alle anderen Programme werden mit einem & am Ende aufgerufen, also in den Hintergrund gestellt. Die Datei .Xdefaults enthält die Ressourcen. Damit können Farben, Schriften und Beschriftungen von Programmen geändert werden.

[« Regulärer Start von X: | X Window starten | Eine kleine Beispielsitzung mit »](#)

Unterabschnitte

- [Varianten des xdm](#)
-

Eine kleine Beispielsitzung mit xdm

Diese Sitzung zeigt am Beispiel auf, wie man sich in einer klassischen X-Umgebung zurecht findet. Wenn Sie die Beispiele im ersten Kapitel nachvollziehen wollen, brauchen Sie eine Terminalsitzung. Die Aufgabenstellung für die Beispielsitzung soll es also sein, sich anzumelden, ein Terminalfenster zu öffnen und zu schließen, sich auszuloggen und zu guter Letzt den Rechner geregelt herunterzufahren. Die gleiche Aufgabe wird später mit anderen Display Managern durchgeführt.

Nach dem Einschalten eines mit xdm installierten Rechners erscheint als Eingabeforderung eine große Dialogbox. Als Überschrift sehen Sie groß den Rechnernamen. Darunter befinden sich die Eingabefelder für login und Passwort. Nach Eingabe der Benutzerkennung, die durch Return abgeschlossen wird, wechselt der Cursor zur Eingabeaufforderung für das Passwort, das blind einzugeben ist. Es erscheint keine Rückmeldung auf dem Bildschirm. Nach einem weiteren Return sind Sie am System angemeldet.

Der Bildschirm kann grundlegend unterschiedlich aussehen, je nachdem, welcher Fenstermanager auf dem System läuft. Mit etwas Glück befindet sich auf dem Bildschirm bereits ein Terminalfenster. In diesem Fall ist es möglich, dass Sie sich ausloggen können, indem Sie dieses Fenster mit `ctrl-D` oder dem Befehl `exit` beenden.

Falls Sie kein Terminalfenster vorfinden, müssen Sie untersuchen, was die Menüs hergeben, die erscheinen, wenn man mit der rechten und dann mit der linken Maustaste auf den Bildhintergrund klickt. X-Terminal oder xterm sind Menüpunkte zum Starten eines Terminalfensters. Durch `ctrl-D` oder den Befehl `exit` schließt das Fenster wieder. Sie können meist über einen anderen Menüeintrag des oben genannten Menüs die Sitzung beenden. Je nach System heißt der Eintrag Quit oder Exit.

Um das System herunterzufahren, dürfen Sie allerdings die Sitzung nicht beenden, sondern bleiben im Terminalfenster, melden sich mit dem Befehl `su` als root an und leiten mit einem der Befehle für den Shutdown (siehe S. shutdown) das Herunterfahren der Maschine ein.

Es ist denkbar, dass der Systemadministrator auch ein Pseudo-Login definiert hat, um das Herunterfahren von Workstations zu ermöglichen.

Varianten des xdm

Neben dem klassischen xdm gibt es inzwischen auch andere Display Manager. So haben KDE mit kdm und GNOME mit gdm jeweils ihren eigenen Login. Praktischerweise reagieren sie auch auf die Konfigurationsdateien des xdm. Eine der auffallendsten Verbesserungen ist die Möglichkeit, an dieser Stelle die Maschine herunterfahren zu können. Für Workstations ist dies eine wichtige Sache. Natürlich ist die Liste der berechtigten Personen begrenzbare.

Wie später noch gezeigt wird, kann man sich mit den Display Managern auch über das Netzwerk einloggen.

« [Grafisches Einloggen: Display Manager](#) | [X Window starten](#) | [X Window benutzen](#) »

X Window benutzen

Das X Window System verhält sich an vielen Stellen etwas anders als andere grafische Oberflächen, sondern ist durch die extrem flexible Anpassbarkeit manchmal von Maschine zu Maschine kaum wieder zu erkennen. So werden Benutzer anderer grafischer Oberflächen manchmal etwas überrascht sein. Einige Elemente und Programme entstammen noch dem Athena Widget Set, das heute etwas archaisch wirkt. Auch die Steuerungsmöglichkeit von Programmen durch Aufrufe von der Shell und die Art der Konfiguration sind auf anderen Oberflächen so nicht üblich. Einige dieser Kuriositäten sind mit den moderneren Desktops verschwunden, andere liegen nur verborgen unter der bunten Oberfläche.

Unterabschnitte

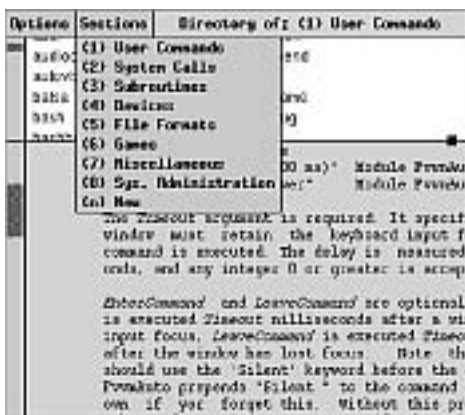
- [Bedienungselemente des Athena Widget Set](#)
- [Der Aufruf von X-Programmen](#)
- [Cut and Paste](#)
- [Terminalfenster xterm](#)
 - [Menüs](#)
 - [Rollbalken](#)
 - [Farbspiele](#)
 - [Start von Applikationen beim Aufruf](#)
 - [Verwandtschaft](#)
- [Weitere praktische Helfer](#)

Bedienungselemente des Athena Widget Set

Einige X-Programme haben sich durch die Jahrzehnte erhalten. Sie wurden mit dem ersten verfügbaren Widget Set, dem Athena Widget Set geschrieben. Die Bedienung ist teilweise ungewöhnlich. Zu den mit dem Athena Widget Set geschriebenen Programmen gehören einige Klassiker.

- [xman]

Das Program zeigt die Manpages. Nach dem Start erscheint erst ein kleines Fenster, das eigentlich nur den Zweck hat, dass man auf den Button »Manual Page« drückt. Erst dann erscheint das eigentliche Hauptfenster von xman. Das Menü besteht nicht aus einer Menüleiste, sondern aus Buttons, denen man eine Menüleiste angehängt hat.



Im Menü Options verbirgt sich der Punkt »Show Both Screens«. Damit teilt sich das Fenster von xman waagerecht. Diese Linie wird rechts durch ein kleines Rechteck unterbrochen, mit dessen Hilfe Sie die Trennlinie verschieben können. Im oberen Teilfenster erscheint eine Liste mit den angebotenen Manpages und im unteren sehen Sie die zuletzt angewählte Manpage. Unter dem Menüpunkt Sections finden Sie die angebotenen Büchereien. xman hat durch die Befehlslisten den Vorteil, dass Sie auch dann noch Hilfe finden, wenn Sie nicht ganz genau wissen, wie der Befehl heißt, den Sie suchen.

- [xterm]

xterm ist eine Terminalemulation. xterm stellt eine Shell wie in der Textoberfläche zur Verfügung. Das Programm xterm wird an anderer Stelle noch ausführlicher betrachtet (siehe S. xterm).

- [xcalc]

Dies ist ein mathematisch wissenschaftlicher Taschenrechner.



- [xpaint]
Ein einfaches Malprogramm für das Erstellen von Grafiken.
- [xedit]
Ein einfacher Texteditor, der aber vielleicht für Einsteiger nicht ganz so ungewohnt ist wie vi oder emacs.

Der Rollbalken erscheint auf der linken Seite des Fensters und dient in erster Linie der optischen Rückmeldung. Er wird durch die drei Maustasten gesteuert. Allerdings ist die Steuerung für Benutzer anderer grafischer Oberflächen etwas ungewohnt.

- [linke Maustaste:]

Blättert weiter. In welcher Schrittgröße vorgegangen wird, hängt davon ab, wo sich der Mauszeiger befindet. Ist er oben, ergeben sich kleine Schritte. Ist er unten, werden die Schritte größer.

- [rechte Maustaste:]

Blättert zurück. Auch hier hängt die Schrittgröße in gleicher Weise von der Position des Mauszeigers im Rollbalken ab.

- [mittlere Maustaste:]

Bewegt den Rollbalkenschlitten an die aktuelle Mausposition. Bei PCs mit nur zwei Maustasten wird die mittlere Taste oft durch das Drücken beider Tasten gleichzeitig emuliert.

Menüs befinden sich beim Athena Widget Set nicht zwingend in der obersten Zeile. Sie sind manchmal gar nicht als solche zu erkennen, da ein Menükopf aus Sicht des Athena Widget Set nichts anderes als ein normaler Button ist.

Der Trennbalken wurde in anderen grafischen Oberflächen erst sehr viel später eingeführt. Dieser Balken hat in der Nähe der rechten Seite eine Verdickung in Rechteckform. Wenn Sie diese mit der Maus anklicken und festhalten, können Sie den Balken verschieben.

« [X Window benutzen](#) | [X Window benutzen](#) | [Der Aufruf von X-Programmen](#) »

Der Aufruf von X-Programmen

Wenn Sie X-Programme von der Shell aus starten, setzen Sie an das Ende der Zeile ein kaufmännisches Und (&). Damit starten Sie es im Hintergrund und haben die Shell anschließend wieder frei.

X-Programme akzeptieren beim Start gleiche Optionen. Der Grund ist, dass diese X-Programme gleich nach dem Start ihre Aufrufparameter an die X-Funktion `XtAppInitialize()` weitergeben, die auch für die Initialisierung des Programms zuständig ist. Diese durchsucht die Parameter nach Optionen, die sich an X richten. Dazu gehören beispielsweise `-fg` oder `-bg` zum Verändern der Farben. Da die Optionen bei allen Programmen von der gleichen Funktion ausgewertet werden, sind die Optionen auch durchgängig für alle X-Programme gültig. Leider gilt das nicht unbedingt für Programme von KDE oder GNOME. Der folgende Aufruf startet `xman` mit einem gelben Hintergrund und roten Buchstaben.

```
xman -bg yellow -fg blue &
```

Neben der Möglichkeit, diese Optionen aus einer Shell den Programmen mitzugeben, haben diese Optionen vor allem ihren Einsatz, um aus Skripten gestartet zu werden, beispielsweise aus der bereits genannten `xinitrc`. Tabelle zeigt einige der wichtigsten Optionen, die Sie unter X verwenden können.

[Standard X-Optionen]L|L Option & Wirkung

- display display & zu verwendender X-Server
- geometry geometry & Startposition und Größe des Fensters
- bg color, -background color & Farbe Fensterhintergrund
- bd color, -bordercolor color & Farbe Fensterrand
- bw pixel, -borderwidth pixel & Breite des Fensterrand
- fg color, -foreground color & Vordergrundfarbe
- fn font, -font font & verwendeter Zeichensatz
- iconic & Programm bei Start als Symbol abstellen
- title string & Der Titel im Schiebebalken

Die kursiv angegebenen Parameter bedürfen noch einer näheren Erläuterung.

- [geometry]
Die Geometrie ist die Ausdehnung des Fensters. Als Parameter wird ohne Leerzeichen hintereinander Breite x Höhe + X-Abstand + Y-Abstand zur linken oberen Ecke angegeben. Beispiel:

```
xcalc -geometry 90x120+60+40 &
```

Die Abstände können auch durch ein Minuszeichen angegeben werden. Dann beziehen sie sich die Abstände auf die untere bzw. rechte Kante. Eine Mischung ist möglich.

- [display]
Der Display ist die exakte Bezeichnung des X-Servers, auf dem das Programm ablaufen soll. Der Aufbau lautet Host: Display.Screen. Genauere Informationen finden Sie auf Seite display.
- [color]

Farben können oft mit ihren englischen Namen verwendet werden, die in /usr/lib/X11/rgb.txt definiert sind. Alternativ gibt es die Möglichkeit, sie als RGB-Werte anzugeben. Beispielsweise als rgb:50/99/99 (siehe S. xcolors).

- [pixel]
Bildpunkte auf dem Bildschirm.
- [font]
Der Zeichensatz kann nach Schriftart, -größe oder -attributen eingegrenzt werden. Nähere Angaben zur Bezeichnung einer Schrift finden Sie auf Seite xfonts.

« [Bedienungselemente des Athena Widget](#) | [X Window benutzen](#) | [Cut and Paste](#) »

Cut and Paste

Unter X gibt es einen extrem schnellen Mechanismus, Texte zwischen Anwendungen per Maus auszutauschen, der sich von anderen grafischen Oberflächen stark unterscheidet. Im ersten Schritt wird der zu kopierende Textbereich markiert. Das wird durch Überstreichen mit der Maus bei gedrückter linker Maustaste erreicht. Es kann immer nur eine Markierung im X geben. Wird etwas anderes in einem anderen Fenster markiert, wird die bisherige Markierung aufgehoben. Durch die mittlere Maustaste wird das Markierte unter dem Mauszeiger als Texteingabe eingefügt. Das betroffene Fenster muss zu diesem Zeitpunkt nicht den Fokus haben, oder anders ausgedrückt, nicht aktiv sein.

Einige Programme, wie etwa StarOffice oder Netscape verwenden das von MS Windows oder MacOS bekannte Verfahren des Cut and Paste über ein so genanntes Clipboard. Dabei wird mit ctrl-C (oder ctrl-insert) die Markierung auf das Clipboard kopiert und mit ctrl-V (oder shift-insert) der Clipboardpuffer wieder eingefügt. Alternativ zum Kopieren kann auch mit ctrl-X (shift-delete) der markierte Bereich entfernt und auf das Clipboard gelegt werden. Der Austausch mit anderen X-Programmen ist zumindest bei StarOffice gewährleistet, da Markierungen außerhalb von StarOffice aus dessen Sicht im Clipboard stehen und das Clipboard von StarOffice vom restlichen X als aktive Markierung angesehen wird.

Soll also ein Text im xterm in ein geöffnetes Dokument von StarOffice gelangen, markieren Sie den Text im xterm und geben im StarOffice an der Zielstelle ctrl-V ein. Um einen Text aus dem StarOffice ins xterm zu bringen, markieren Sie ihn im StarOffice, drückt ctrl-C und können ihn nun im xterm durch Drücken der mittleren Maustaste einfügen.

Unterabschnitte

- [Menüs](#)
 - [Rollbalken](#)
 - [Farbspiele](#)
 - [Start von Applikationen beim Aufruf](#)
 - [Verwandtschaft](#)
-

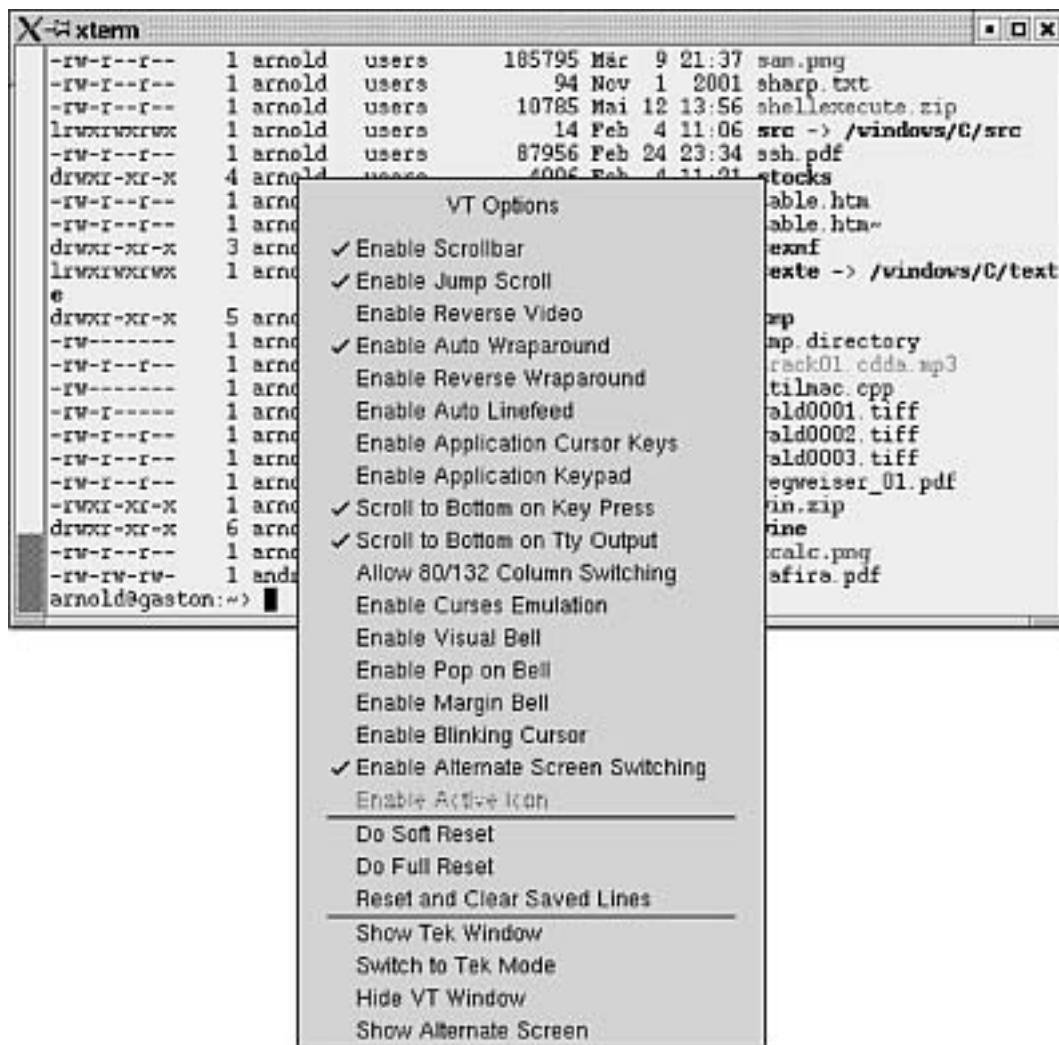
Terminalfenster xterm

Terminalemulationen haben eine besondere Bedeutung unter X. Da die Shell unter UNIX so leistungsfähig ist, gibt es einige Dinge, die man mit ihr einfacher realisieren kann als mit einer grafischen Benutzeroberfläche. Aus diesem Grund benutzt der erfahrene UNIX-Anwender gern beides nebeneinander, ohne Ideologieprobleme zu haben. Die Existenz und Bedeutung von xterm ist kein Zeichen für das Versagen der GUI, sondern ein Indiz für die Leistungsfähigkeit der Shell.

Sie können das Fenster von xterm jederzeit vergrößern oder verkleinern. Die friedliche Koexistenz von GUI und textorientierten Programmen zeigt sich daran, dass die meisten UNIX-Anwendungen die Größenänderungen direkt berücksichtigen und umsetzen. Wenn man in einem xterm-Fenster vi laufen lässt, passt vi im Betrieb die Größe des Arbeitsbereichs automatisch an.

Menüs

Auch xterm besitzt Menüs, allerdings nicht in einer Menüleiste. Sie erreichen das Menü durch Halten der ctrl-Taste und den Druck einer Maustaste. Mit der linken Taste erscheinen die Hauptoptionen, mit der mittleren die VT-Optionen und mit der rechten Maustaste die VT-Fonts.



Das linke Menü ermöglicht das Senden von Signalen und die Einstellung einiger Besonderheiten wie der Funktionstastenbelegung. Die wichtigste Funktion der mittleren Maustaste ist das Zuschalten des Rollbalkens, der auf der linken Seite erscheint. Hier wird offenkundig, dass xterm auf dem Athena Widget Set basiert. Die rechte Maustaste bietet die Möglichkeit, unterschiedlich große Zeichensätze zu verwenden.

Rollbalken

Wie bereits erwähnt wurde, können Sie den Rollbalken durch das Menü einschalten, das sich durch ctrl-mittlere Maustaste erreichen lässt. Gewöhnungsbedürftig ist allerdings die Benutzung. Der Rollbalken befindet sich ungewohnt am linken Fensterrand. Durch Linksklick mit der Maus auf irgendeine Stelle im Rollbalken geht es weiter nach unten und durch Rechtsklick zurück. Dabei werden die Schritte, in denen geblättert wird um so größer, je weiter unten Sie klicken. Um den Schieber so zu bewegen, wie Sie das von anderen Rollbalken kennen, wird die mittlere Maustaste verwendet. Vor allem bietet der Rollbalken die Möglichkeit, schnell durchgelaufene Inhalte in Ruhe anzuschauen. Damit ist es nur noch selten nötig, die Ausgabe durch more zu schicken.

Farbspiele

Unter X-Window können die verwendeten Farben der meisten Anwendungen bei ihrem Aufruf konfiguriert werden. Beispiel:

```
xterm -fg blue -bg yellow &
```

Dieser Befehl startet ein neues Terminalfenster mit blauen Buchstaben auf gelbem Grund. Durch die Farben können Sie leicht ein wenig Übersicht in einen Bildschirm voller Fenster bringen. So könnten Sie Fenster, in denen Sie als root arbeiten wollen, mit einem roten Hintergrund versehen. Es ist auch hilfreich, wenn Sie Fenster, mit denen Sie in verschiedenen Verzeichnissen oder auf verschiedenen Rechnern arbeiten, unterschiedlich einfärben können.

Start von Applikationen beim Aufruf

Mit der Option `-e` können beim Start eines `xterm` direkt Programme innerhalb des Terminals gestartet werden. Beispielsweise kann auf diese Weise eine andere Shell gestartet werden. Sie können aber auch ein Fenster mit einem `vi` oder mit `elm`, einem E-Mail Programm, starten. Sobald das Programm beendet ist, schließt auch das `xterm`. `-e` muss immer am Ende der Optionen stehen, da so auch dem mitgestarteten Programm Parameter übergeben werden können.

```
xterm -e vi unixx11.htm &
```

So wird in dem Fenster der `vi` mit der Datei `unixx11.htm` gestartet. Sobald `vi` verlassen wird, schließt auch das Fenster.

Verwandtschaft

Neben dem klassischen `xterm` gibt es unter fast jeder X11-Abart eine andere Terminalversion. Einige Beispiele:

- [xterm Derivate]L|L Plattform & Terminalprogramm
- Sun Solaris CDE & `dtterm`
- HP-UX & `hpterm`
- Linux KDE & `kvt`, `konsole`
- GNOME & `gnome-terminal`
- MacOS X & `Terminal.app`

`xterm` wirkt neben seinen Verwandten oft etwas karg. Allerdings hat `xterm` den Vorteil, dass es unter X immer zur Verfügung steht, und sich auf allen Umgebungen gleich verhält.

« [Cut and Paste](#) | [X Window benutzen](#) | [Weitere praktische Helfer](#) »

Weitere praktische Helfer

Das Programm `xkill` verwandelt den Mauszeiger in einen Totenkopf, denn es beendet das Programm, in dessen Fenster als nächstes geklickt wird.

Mit dem Programm `xwd` können Sie eine so genannte Hardcopy vom Bildschirm herstellen. Das Programm wird gestartet, dann markieren Sie mit der Maus den zu kopierenden Bereich. Ein Kontrollpiep meldet, dass der Bereich erfasst ist. Die Ausgabe wird nach `stdout` geschrieben oder durch die Option `-out` umgeleitet. Das Format dieser Datei nennt sich XWD und ist ein originäres X Window Format. Zum Betrachten dieser Bilder gibt es das Programm `xwud`, aber auch moderne Programme wie `gimp` können mit diesem Format umgehen.

```
silver> xwd -out huhu.xwd
silver> xwud -in huhu.xwd &
```

Zusätzliche, kleinere Komfortprogramme werden ebenfalls mitgeliefert. `xclock` ist eine Uhr, `xcalc` ein kleiner Taschenrechner. Mit `xload` kann man sich die Belastung der Maschine ansehen. `xbiff` meldet, wenn E-Mail eingetroffen ist.

[« Terminalfenster xterm | X Window benutzen | Konfigurieren »](#)

Konfigurieren

Auch unter dem X Window System lassen sich die meisten Konfigurationen über Textdateien erreichen, die Sie mit einem beliebigen Editor bearbeiten können.

Unterabschnitte

- [Farbbeschreibung](#)
 - [Schriften](#)
 - [Bitmaps](#)
 - [Ressourcen](#)
 - [Konfiguration des Fenstermanagers](#)
 - [Fokus und Z-Anordnung](#)
-

Farbbeschreibung

Unter X-Window können die verwendeten Farben der meisten Anwendungen bei ihrem Aufruf konfiguriert werden. Oben war schon einmal als Beispiel ein xterm mit blauen Buchstaben auf gelbem Grund gezeigt worden:

```
xterm -fg red -bg yellow &
```

Die verwendeten Farbnamen red und yellow werden in der Datei rgb.txt definiert. Die Datei finden Sie im lib-Verzeichnis des X, beispielsweise unter /usr/lib/X11. Hier werden die RGB-Umsetzungen der bekannten Farbnamen abgestellt. Aus diesem Grund können Sie viele Farben über ihre englischen Namen ansprechen. In den meisten Fällen können Sie auch die Attribute light oder dark verwenden, wie beispielsweise lightcyan oder darkblue.

255	250	250	snow
248	248	255	Ghostwhite
245	245	245	whiteSmoke
255	228	181	moccasin
255	248	220	cornsilk
255	255	240	ivory
255	250	205	LemonChiffon
240	255	255	azure
240	248	255	alice blue
240	248	255	AliceBlue
230	230	250	lavender
255	255	255	white
0	0	0	black
47	79	79	DarkSlateGray
105	105	105	DimGray

Sie können statt dem Farbnamen auch direkt die Definition der RGB"-Farben verwenden. Dazu wird das Schlüsselwort rgb vorangestellt. Die Syntax lautet:

rgb:Rotanteil/Grünanteil/Blauanteil

Die Werte für die jeweiligen Anteile werden hexadezimal, ein- bis vierstellig angegeben. Das folgende Beispiel startet xterm mit knallrotem Hintergrund:

```
silver> xterm -bg rgb:F/0/0 &
```

« [Konfigurieren](#) | [Konfigurieren](#) | [Schriften](#) »

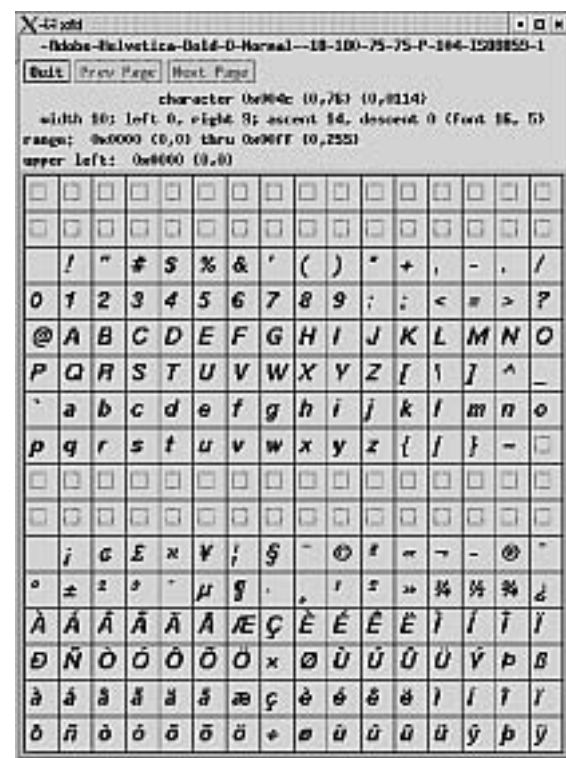
Schriften

Mit dem Kommando `xlsfonts` können Sie sich alle Zeichensätze ansehen, die X zur Verfügung stehen. Die Liste dürfte im Normalfall recht lang sein. Die Namen der Schriften enthalten auch technische Details, die für einen Setzer oder Grafiker von besonderer Bedeutung sind, die man bei der Angabe eines Zeichensatzes für Standardprogramme ungern mit angeben möchte. Hier hilft X mit der Möglichkeit, bei der Auswahl von Schriften den Stern als Platzhalter zu verwenden, wie Sie das bei der Dateiauswahl der Shell kennen. Der `*` steht wie dort für beliebig viele Zeichen. Wenn ein Kommando mit einem Stern für die Schriften in der Shell aufgerufen wird, muss der entsprechende Parameter in Anführungszeichen stehen, damit die Shell nicht versucht, ihn aufzulösen.

Zum Betrachten eines Zeichensatzes dient das Programm `xfd`. Der gewünschte Zeichensatz wird mit der Option `-fn` angegeben. Mit diesem Tool können Sie sich nicht nur das Aussehen der Zeichensätze anzeigen lassen. Sie können sich auch die Größe jedes einzelnen Zeichens ansehen. Dazu können Sie erkennen, ob die Schriftbezeichnung, die Sie beim Aufruf von `xfd` verwendet haben, auch die Schrift anzeigt, die Sie sich vorgestellt haben.

```
gaston > xfd -fn "*helvetica*-180-*"
gaston > xterm -fn "*helvetica*-180-"
```

Im ersten Schritt wird der Font mit dem Namen `helvetica` angezeigt. In der zweiten Zeile sehen Sie, dass man den gleichen Parameter bei den meisten X-Programmen verwenden kann, um deren Schrift zu bestimmen.



Die oben angegebene Schrift heißt korrekt auf meinem System

-Adobe-Helvetica-Bold-O-Normal-18-180-75-75-P-104-ISO8859-1

Obwohl nach »helvetica« gesucht wurde, wurde »Helvetica« gefunden. Man sieht deutlich, dass bei den Schriftnamen Klein- und Großschreibung nicht relevant ist. Der Name der Schrift enthält mehrere Informationen über die Schrift selbst.

- [Adobe] Der Hersteller der Schrift.
- [Helvetica] Die Schriftfamilie. Daneben gibt es noch Schriften wie Times oder Courier. Übrigens ist die Schrift Arial fast nur unter MS Windows verfügbar. Darum sollte man sie nicht unbedingt im Internet verwenden oder wenn man seine Dokumente mit einem UNIX- oder Macintosh-Benutzer austauscht.
- [Bold] Die Angabe bezeichnet, wie fett die Schrift ist. Wer auf Diät ist, sollte statt bold (fett) lieber light oder medium verwenden.
- [O] Das O steht für oblique und bezeichnet die Neigung der Schrift. Alternativ steht hier R für roman oder I für italic (kursiv).
- [18] Die Zeichenhöhe in Pixeln
- [180] Der Schriftgrad in zehntel Punkt. Diese Schrift ist also eine 18 Punkt Schrift. Ein Punkt ist eine Maßeinheit aus dem Satzgewerbe und entspricht etwa 1/72 Zoll.
- [75-75] Die horizontale und vertikale Auflösung des Gerätes, für das die Schrift entwickelt wurde. In diesem Fall ist die Auflösung des Zielgerätes 75 dpi in der Breite und in der Höhe. Ein Bildschirm liegt normalerweise etwa in der Auflösung zwischen 75 und 100 dpi.
- [P] Das P steht für proportional, bei Schriften mit festem Zeichenabstand wird ein M für monospaced verwendet.
- [ISO8859-1] Das ist der Zeichensatz, der verwendet wird. Es handelt sich um das »Latin Alphabet No. 1«, dass mehrere europäische Sprachen unterstützt.

Wenn Sie sich eine Schrift der Familie Times mit mittlerer Dicke ohne Neigung in der Größe 12 Punkt ansehen wollen, würden Sie folgende Bezeichnung verwenden:

```
xfd -fn "*-Times-Medium-R-*-120-*"  
xlsfonts "*-Times-Medium-R-*-120-*"
```

Der Aufruf von `xfd` zeigt den ersten passenden Zeichensatz an. Der Aufruf von `xlsfonts` zeigt an, welche passenden Schriften installiert sind.

Wer seinem System Schriften hinzufügen will, findet weitere Informationen auf den Manpages von `mkfontdir`, `xset` und dem Font Manager `xfs`.

« [Farbbeschreibung](#) | [Konfigurieren](#) | [Bitmaps](#) »

Bitmaps

X liefert das Programm `bitmap` zum Erstellen von Icons mit. Damit können Sie Mauszeiger oder Symbole für Programme gestalten. Die Bedienung von `bitmap` ist intuitiv. Sie bekommen ein Raster in dem Sie Ihr Icon malen können. Mit der linken Maustaste setzen Sie einen Punkt, mit der rechten Taste löschen und mit der mittleren invertieren Sie ihn.

Das Icon wird als Textdatei abgespeichert, genauer gesagt findet man darin sogar den C-Quelltext. Mit den Programmen `atobm` bzw. `bmtoa` kann die Bitmap konvertiert werden.

Das Programm `xsetroot` bestimmt den Bildschirmhintergrund. Mit der Option `-bitmap` können Sie auch Icons als Hintergrund verwenden. Sie werden dann gekachelt. Die Option `-solid` Farbe wird den Hintergrund einfärben.

« [Schriften](#) | [Konfigurieren](#) | [Ressourcen](#) »

Ressourcen

Unter Ressourcen versteht man unter X Vorgabedateien, in denen Sie definieren können, welche Vorgabewerte ein Programm bekommt. Je nach Programm können dort Programmparameter angepasst werden. Vor allem können Sie die Eigenschaften bezüglich des grafischen Erscheinungsbildes verändern.

Es gibt verschiedene Dateien, in denen die Ressourcen definiert werden können.

- [/usr/lib/X11/app-defaults/Klassenname]
Diese Datei enthält für die Applikation der Klasse Klassenname die für das System geltenden Ressourcen.
- [\$HOME/Klassenname]
Hier stehen die vom Anwender spezifizierten Ressourcen für die Applikation der Klasse Klassenname.
- [\$HOME/.Xdefaults]
Hier stehen die vom Anwender definierten Ressourcen
- [\$HOME/.Xdefaults-host]

Als Beispiel soll die Uhr xclock mit rotem Hintergrund und blauen Zeigern erscheinen. Dazu legen Sie im eigenen Heimatverzeichnis die Datei XClock an und schreiben folgendes hinein:

```
*foreground: blue
xclock*background: red
```

In der Datei XClock ist die Angabe des Klassennamens XClock in der zweiten Zeile natürlich redundant. In der Datei .Xdefaults dagegen wird die Angabe auf das Programm xclock beschränkt.

Nach dem Start von xclock erscheint eine Uhr mit rotem Hintergrund und blauen Minutenunterteilungen. Nennen Sie die Datei XClock in xclock um, passiert das nicht mehr. Der Grund liegt darin, dass XClock der Klassenname des Programms xclock ist. Sofort stellt sich die Frage, wie Sie diesen Namen erfahren. Da kann das Programm xprop helfen. Nach dem Start aus einer Shell verwandelt sich der Cursor in ein Kreuz. Klicken Sie nun ein Fenster an, erhalten Sie diverse Informationen. Unter WM_CLASS finden Sie zwei Namen. Der zweite ist normalerweise der Klassenname. Zuerst wurde xclock in den Hintergrund gestartet. Nach dem Aufruf von xprop ändert sich der Mauscursor zum Fadenkreuz. Durch das Anklicken des Fensters von xclock erscheint die untenstehende Ausgabe.

```
gaston> xclock &
[1] 3094
gaston> xprop
_NET_WM_ICON_GEOMETRY(CARDINAL) = 980, 602, 38, 20
WM_PROTOCOLS(ATOM): protocols WM_DELETE_WINDOW
_NET_WM_DESKTOP(CARDINAL) = 1
_KDE_NET_WM_FRAME_STRUT(CARDINAL) = 4, 4, 20, 8
_NET_WM_VISIBLE_NAME(UTF8_STRING) = 0x78, 0x63, 0x6c, 0x6f,...
WM_STATE(WM_STATE):
    window state: Normal
    icon window: 0x0
SM_CLIENT_ID(STRING) = "11c0a86d90000102321350600000010070034"
WM_CLIENT_LEADER(WINDOW): window id # 0x4000009
```

```
WM_LOCALE_NAME(STRING) = "C"
WM_CLASS(STRING) = "xclock", "xclock"
WM_HINTS(WM_HINTS):
    Client accepts input or input focus: False
    Initial state is Normal State.
    bitmap id # to use for icon: 0x4000001
    bitmap id # of mask for icon: 0x4000003
WM_NORMAL_HINTS(WM_SIZE_HINTS):
    program specified size: 164 by 164
    window gravity: Northwest
WM_CLIENT_MACHINE(STRING) = "gaston"
WM_COMMAND(STRING) = { "xclock" }
WM_ICON_NAME(STRING) = "xclock"
WM_NAME(STRING) = "xclock"
gaston>
```

Die nächste Frage ist, woher man weiß, dass man mit background den Hintergrund und mit foreground die Minutenstriche färbt. Die einfachste Lösung ist ein Blick in die Manpages. Dort stehen normalerweise die Attribute dokumentiert, die man per Ressource verändern kann.

Zu guter Letzt stellt sich die Frage nach den Sternen. Auch hier in den Ressourcen verwendet man den Stern für beliebig viele Zeichen. Die obere Zeile gibt an, dass alles, was auf foreground endet, blau sein soll. Die darunter liegende Zeile schränkt die angesprochenen Objekte stärker ein. Zuerst muss XClock erscheinen, dann irgend etwas und zu guter Letzt background. Da beides in der Datei XClock steht, kann es nur xclock betreffen. Wenn beides aber in der .Xdefaults stünde, wäre die Einschränkung auf XClock schon wichtig.

Jedes einzelne Widget kann verändert werden. Um mehrere Widgets in einer Zeile zu definieren, kann der Stern verwendet werden. Eine solche Datei kann beispielsweise aussehen:

```
bitmap*Dashed: off
XTerm*cursorColor: gold
XTerm*multiScroll: on
XTerm*jumpScroll: on
XTerm*reverseWrap: on
XTerm*curses: on
XTerm*Font: 6x10
XTerm*scrollBar: on
XTerm*scrollbar*thickness: 5
XTerm*multiClickTime: 500
XTerm*charClass: 33:48,37:48,45-47:48,64:48
XTerm*cutNewline: off
XTerm*cutToBeginningOfLine: off
XTerm*titeInhibit: on
XTerm*ttyModes: intr ^c erase ^? kill ^u
XLoad*Background: gold
XLoad*Foreground: red
XLoad*highlight: black
XLoad*borderwidth: 0
emacs*Geometry: 80x65-0-0
emacs*Background: rgb:5b/76/86
emacs*Foreground: white
emacs*Cursor: white
emacs*BorderColor: white
emacs*Font: 6x10
xmag*geometry: -0-0
xmag*borderColor: white
```

Jeder Benutzer kann in seinem Verzeichnis unter dem Namen .Xdefaults eine Datei anlegen, die bei Start automatisch zu den Systemressourcen hinzugeladen wird.

« [Bitmaps](#) | **Konfigurieren** | [Konfiguration des Fenstermanagers](#) »

Konfiguration des Fenstermanagers

Die Startupdatei liegt bei den meisten Fenstermanagern im Verzeichnis `/usr/lib/X11`. Darin befindet sich ein Verzeichnis mit dem Namen des Fenstermanagers, in dem sich wiederum eine `system.XXwmrc` findet. Hierin werden Menüs, Tastenzuordnungen und Mausaktionen definiert.

```
Menu DefaultRootMenu

    "Root Menu"          f.title
    "New window"         f.exec "xterm -e /bin/bash &"
    "Shuffle Up"         f.circle_up
...
    "Restart fvwm"       f.restart "fvwm"
    "Quit..."          f.quit_mwm
```

Der Ausschnitt stammt aus der Konfigurationsdatei eines `twm`. Diese Syntax gilt aber leider nicht für alle Fenstermanager. Entsprechend macht es wenig Sinn, an dieser Stelle das Thema besonders zu vertiefen. Informationen über die Syntax und die Möglichkeiten des jeweiligen Fenstermanagers finden sich in dessen Manpage.

[« Ressourcen](#) | [Konfigurieren](#) | [Fokus und Z-Anordnung](#) [»](#)

Fokus und Z-Anordnung

Unter dem Fokus versteht man bei grafischen Oberflächen das Element, das auf die Tastatureingabe anspricht. Man kann auch vom aktiven Fensterelement sprechen. Unter der Z-Anordnung versteht man die Rangfolge der Fenster übereinander. Vom Macintosh und auch später von MS Windows sind PC-Benutzer es gewohnt, dass das Fenster, das als letztes angeklickt wurde, nach vorn kommt und damit in der Z-Anordnung die erste Position einnimmt und den Fokus erhält. Dieses Verhalten ist bei X konfigurierbar. Standardmäßig wurde X immer so ausgeliefert, dass das Fenster oder auch Element unter dem Mauszeiger den Fokus erhielt. Das bedeutet nicht zwingend, dass dieses Fenster auch nach »vorn« kommt. Das geschieht durch Anklicken oder durch eine einstellbare Verzögerungszeit.

Sie müssen auf solchen Systemen ein wenig aufpassen, wo Sie den Mauszeiger auf dem Bildschirm hinschieben. Hat man sich an diese Technik gewöhnt, ist das Arbeiten mit mehreren Fenstern aber sehr viel schneller.

Unter Motif wurde erstmals das vom Mac gewohnte Verfahren als Standard verwendet. Man spricht vom »click to focus«. Die Behandlung des Fokus wird bei Motif in den Ressourcen festgelegt. Dort stehen drei Variablen für Mwm zur Verfügung, mit denen das Fokusverhalten bestimmt wird. Wenn Sie ein anderes Verhalten haben möchten, als das vom System vorgegebene, so können Sie dies durch Einträge in die Datei ~/.Xdefaults umstellen. Die folgende Einstellung erzeugt die klassische Art des Fokus unter der Maus und wartet 250ms bis das Fenster nach vorn springt.vgl OSF/Motif User's Guide. Open Software Foundation. Revision 1.2. 1992. pp. 5-17,5-18.

```
Mwm*keyboardFocusPolicy: pointer
Mwm*focusAutoRaise: True
Mwm*autoRaiseDelay: 250
```

Der Grund, warum man eine Verzögerung beim Hochkommen der Fenster einbaut, ist, dass ansonsten bei einer schnellen Bewegung mit der Maus über den Bildschirm alle Fenster hin- und herspringen. Die Standardeinstellung der FocusPolicy heißt explicit.

Auch unter den modernen Desktops wie KDE, GNOME und CDE lässt sich die Fokusstrategie auf diese Weise einstellen. Sie finden sie im KDE Kontrollzentrum bzw. im CDE Window Style Manager jeweils unter Window Behavior.

Desktops

Der Fenstermanager legt unter X die gleichförmige Bedienung und das Aussehen fest. Der nächste Schritt, höheren Bedienungskomfort zu schaffen, ist der Desktop. Hier rufen Programme sich gegenseitig auf. Beispielsweise startet das Mailprogramm den Entpacker, wenn ein Benutzer das ZIP-File sehen will, das im Anhang der E-Mail steht. Und der Entpacker ruft wiederum den Bildbetrachter auf, wenn der Benutzer eine JPEG-Datei Eine JPEG-Datei ist eine Bilddatei, die in ihrer Art der Komprimierung für Fotos optimiert ist. anwählt.

Die meisten UNIX Desktops besitzen ein so genanntes Panel. Ein Panel ist ein Balken, auf dem Bedienelemente angebracht sind. Teilweise bewegen sie sich frei auf dem Bildschirm, teilweise sind sie am Rand des Bildschirms platziert. Das Panel ist die Schaltzentrale des Desktops.

Bereits unter Motif gab es die ersten Ansätze, die Einstellungen auch über grafische Werkzeuge zu konfigurieren. Bei den Desktops wurde das vervollkommenet. So besitzen sie teilweise Kontrollzentren, in denen die Einstellungsdialoge der Oberfläche zusammengefasst sind.

Das gleichförmige Erscheinungsbild eines Desktop wird auch dadurch erreicht, dass bei gleichen Aufgaben immer die gleichen Wege in den unterschiedlichen Applikationen beschritten werden. Zu diesem Zweck sind Standarddialoge ein wichtiger Schritt. Typische Aufgaben, die immer wiederkehren, ist die Dateiauswahl, die Schriftartenwahl, der Farbdiallog und das Drucken. Die Standarddialoge sind bereits mit einigen Widget Sets wie beispielsweise Motif eingeführt worden.

Zunächst wurde CDE auf der Basis von Motif entwickelt. CDE findet sich auf den Systemen von Sun, Hewlett Packard und IBM. Unter Linux ist Motif nicht etabliert. So entstanden hier zwei Ansätze KDE und GNOME, die der gleichen Idee folgten, aber sich in ihrem Erscheinungsbild unterscheiden.

Völlig anders ist der Desktop des MacOS X, der nicht auf X Window aufbaut, sondern eine eigenen Grafik-Engine (Quartz) und einen eigenen Window-Manager benutzt. Hier wurden viele Eigenschaften des alten MacOS übernommen, um dem Benutzer die Gewöhnung an den neuen UNIX-Unterbau zu vereinfachen.

Unterabschnitte

- [CDE](#)
 - [Panel](#)
 - [Das Menü](#)
 - [Werkzeuge](#)
 - [Eine kleine Beispielsitzung](#)
 - [Mama Motif](#)
- [KDE](#)
 - [Das Panel](#)
 - [Icons](#)

- Konqueror, Dateimanager und Browser
 - Der Stylemanager
 - Eine kleine Beispielsitzung
 - KDE-Terminal: konsole
 - KDE-Programme
 - GNOME
 - Das Panel
 - Icons
 - Dateimanager Nautilus
 - Look and Feel
 - Eine kleine Beispielsitzung
 - Hinter den Kulissen
 - GNOME-Programme
 - Der Wettstreit der freien Desktops
 - MacOS X
 - Eine kleine Beispielsitzung
 - Programme für MacOS X
-

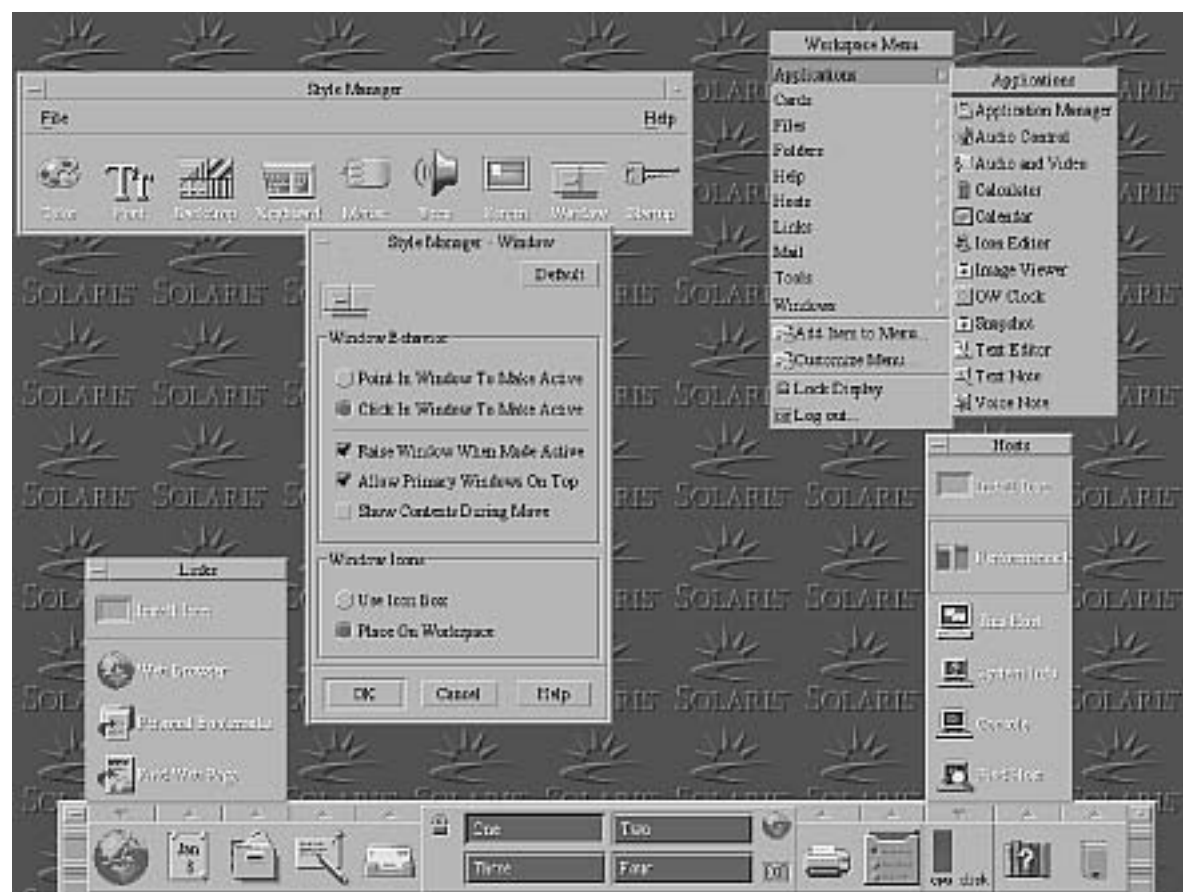
« Fokus und Z-Anordnung | **Das X-Window System** | CDE »

Unterabschnitte

- [Panel](#)
- [Das Menü](#)
- [Werkzeuge](#)
- [Eine kleine Beispielsitzung](#)
- [Mama Motif](#)

CDE

CDE (Common Desktop Environment) ist aus einer Zusammenarbeit einiger großer UNIX-Hersteller entstanden. Mit Motif ist die Bedienung der X Oberfläche bereits an dem normalen Anwender orientiert worden. Hierauf baut CDE auf und liefert ein Panel und viele grafische Werkzeuge, mit denen die Anpassung an die individuellen Bedürfnisse auch ohne das Verständnis der Ressourcen möglich ist.



In der Abbildung sehen Sie unten das Panel mit seinen Startflächen und in der Mitte den Umschalter zwischen den vier Bildschirmen. Als Beispiel sind zwei Menüs aufgeklappt, die weitere Startflächen enthalten. Links oben ist der Stilmanager zu sehen. Darunter der Dialog zur Einstellung der Fokusstrategie. Rechts oben ist mit Hilfe der rechten Maustaste auf dem Hintergrund ein Menü aufgeklappt worden. Von dort erreichen Sie diverse Programme.

Panel

CDE hat als auffallendstes Merkmal ein Panel, in dessen Mitte ein Umschalter zwischen vier virtuellen Bildschirmen sitzt. Den Bildschirmen können durch zweimaliges Klicken Namen zugeordnet werden. Die geöffneten Fenster sind dem jeweiligen Bildschirm zugeordnet und verschwinden beim Umschalten. Auf diese Weise kann man die Fensterflut selbst bei kleineren Monitoren im Griff behalten.

Links und rechts befinden sich je fünf Buttons mit den wichtigsten Funktionen. Gleichzeitig haben diese Buttons Anzeigecharakter. So zeigt der eine die Uhrzeit und der andere die Systemlast. Den Schaltflächen zugeordnet sind kleine, mit Pfeilen nach oben markierte Tastflächen, hinter denen sich jeweils ein Menü verbirgt.

Direkt um den Umschalter herum befinden sich kleinere Tastenfelder für das Abmelden und das kurzfristige Sperren des Bildschirms.

Das Menü

Mit einem Klick der rechten Maustaste auf den Hintergrund erreichen Sie das Arbeitsplatzmenü. In diesem befinden sich Kategorien von Programmen, die weitere Untermenüs mit den eigentlichen Programmen aufklappen, wenn Sie sie anklicken.

Ein Menüpunkt öffnet ein Fenstermenü, mit dem Sie Fenster rotieren, aber auch wie mit `xkill` vernichten können.

Zwei Menüpunkte dienen der Konfiguration des Menüs. So können Sie auch selbst Menüpunkte einbauen. Zu guter Letzt finden Sie die Punkte zum Sperren des Bildschirms und zum Ausloggen.

Werkzeuge

Der Dateimanager ermöglicht den fensterorientierten Zugang auf Dateien. Auf diesem Wege sind die grundlegenden Dateimanipulationen wie Kopieren, Umbenennen und Löschen möglich.

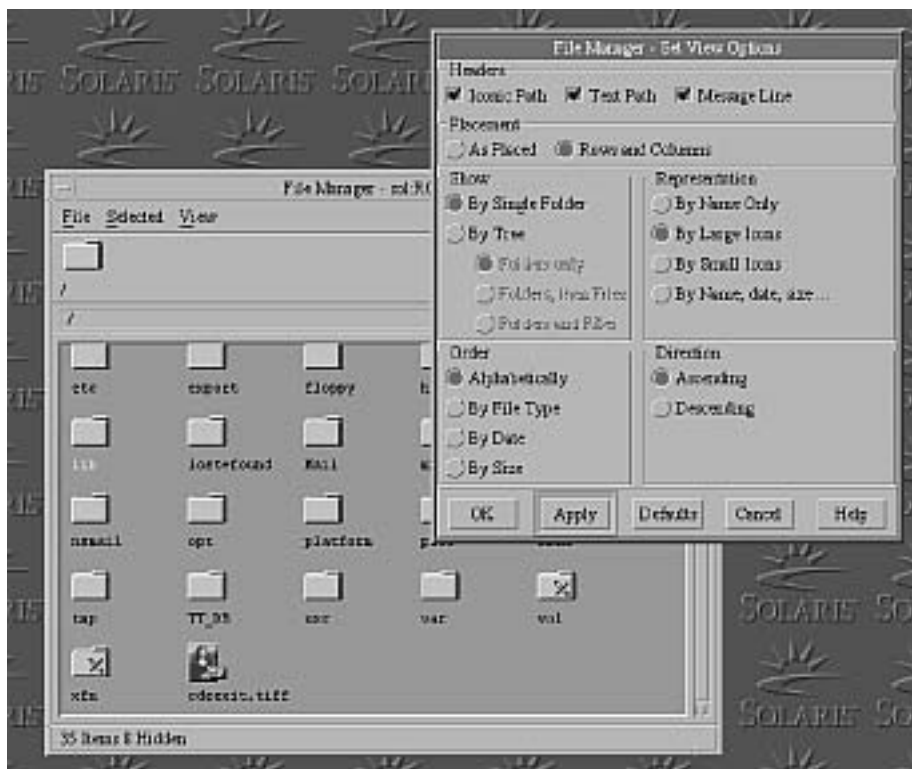


Abbildung zeigt im Hintergrundfenster den Dateimanager in der üblichen Symboldarstellung der Dateien und Ordner. Unter dem Menü wird der Pfad in aufeinanderfolgenden Ordnersymbolen dargestellt. In der darunterliegenden Eingabezeile kann das Arbeitsverzeichnis auch direkt eingegeben werden. Rechts im Vordergrund sieht man den Dialog zur Einstellung der Darstellung. So ist neben der Symbol- auch eine Baum- oder Textdarstellung möglich. Die grundlegenden Sortierkriterien sind einstellbar.

Der Applikationsmanager verwaltet die installierten CDE-konformen Applikationen. Er bietet eine Liste der ausführbaren Programme an und verwaltet die zugehörigen Icons.

Der Sitzungsmanager verwaltet die Sitzungen eines Benutzers. Insbesondere meldet er den CDE-konformen Applikationen das Ende einer Sitzung und nimmt einen String entgegen, der es ermöglicht, die Applikation im gleichen Zustand wieder zu starten, wenn der Anwender sich erneut einloggt.

Mit dem Stilmanager können Sie die Grundeinstellungen über grafische Oberflächen verwalten, statt die Ressourcen direkt zu manipulieren.

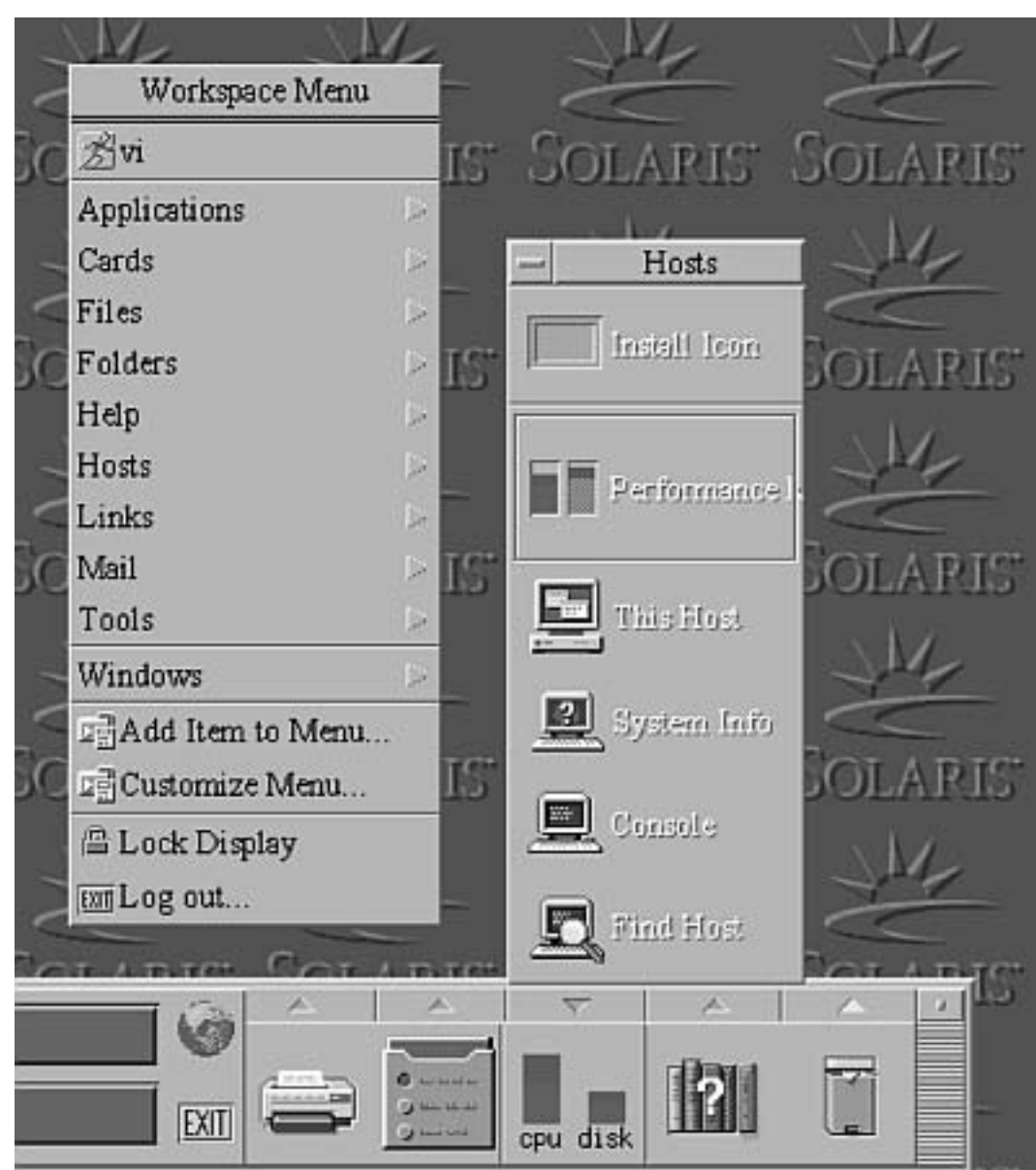
Der Druckmanager macht dem Anwender die Druckerwarteschlangen zugänglich. Ebenfalls befinden sich im Grundpaket ein Kalender, ein Taschenrechner und einige weitere kleinerer Programme.

Eine kleine Beispielsitzung

Die Aufgabenstellung für die Beispielsitzung ist es, sich anzumelden, ein Terminalfenster zu öffnen und zu schließen, sich auszuloggen und zu guter Letzt den Rechner geregelt herunterzufahren.

Nach dem Einschalten eines mit CDE installierten Rechners erscheint als Eingabeforderung eine einzelne Zeile für den Benutzernamen. Nach Eingabe der Benutzerkennung, die durch Return abgeschlossen wird, erscheint eine Eingabeaufforderung für das Passwort, das blind einzugeben ist. Es gibt keine Rückmeldung auf dem Bildschirm.

Bei älteren Versionen der CDE sehen Sie auf einem der Panelknöpfe einen kleinen Computer mit Monitor abgebildet. Wenn Sie diesen anklicken, erscheint ein Terminalfenster. Bei den neueren Versionen, aber auch wenn jemand das Panel seinen eigenen Wünschen angepasst hat, wird dieser Knopf nicht mehr oben auf liegen. Um die Konsole zu finden, müssen Sie auf die Pfeile oberhalb der Knöpfe klicken. Dadurch fährt ein Menü aus, das diesen kleinen Computer mit der Beschriftung Konsole hat. Bei Solaris 8 ist dies der dritte Knopf von rechts, auf dem sich das CPU-Meter befindet, wie in Abbildung zu sehen ist.



Sie melden sich vom CDE ab, indem Sie den Knopf »EXIT« auf dem Panel anklicken. Es gibt noch einmal eine Rückfrage und dann landen Sie im Anmeldebildschirm. Alternativ können Sie das Arbeitsplatzmenü über die rechte Maustaste aufklappen und dort den Menüpunkt zum Abmelden anwählen.

Um das System herunterzufahren, dürfen Sie sich allerdings nicht ausloggen, sondern bleiben in dem Fenster, melden sich mit dem Befehl `su` als `root` an und leiten mit einem der Befehle für den Shutdown (siehe S. shutdown) das Herunterfahren der Maschine ein.

Es ist aber denkbar, dass der Systemadministrator ein Pseudo-Login definiert hat, um das

Herunterfahren von Workstations zu ermöglichen (siehe S. shutdownuser).

Mama Motif

CDE basiert auf dem OSF/Motif. OSF ist die Open Software Foundation, ein Zusammenschluss mehrerer UNIX-Hersteller, angefangen mit DEC, IBM und Hewlett Packard. Einige Zeit versuchte Sun, sein OpenWindows als übergreifenden Standard zu platzieren, wechselte dann aber auch zu Motif und CDE.

Motif basierte auf Ideen IBMs des Common User Access (CUA), der damals in Zusammenarbeit mit Microsoft Grundlage des Oberflächendesigns für OS/2 war.

Wenn Sie unter Motif oder einem der Fenstermanager ein Fenster minimieren, entsteht ein Icon auf dem Bildschirm. Durch einen Doppelklick auf diese Icons wird die ursprüngliche Größe wieder hergestellt. Die anderen Desktops verwenden Icons als Startschalter für Applikationen, wie sie von Macintosh und MS Windows her bekannt sind. Der Unterschied liegt darin, dass im ersten Fall der Prozess bereits läuft, aber das Fenster verkleinert ist. Im zweiten Fall wird das Programm erst durch den Doppelklick gestartet. Allerdings ist es unter CDE auch möglich, Icons auf dem Desktop abzulegen, die in diesem Sinne agieren. Sie sind etwas kleiner als die minimierten Fenster.

« [Desktops](#) | [Desktops](#) | [KDE](#) »

Unterabschnitte

- [Das Panel](#)
- [Icons](#)
- [Konqueror, Dateimanager und Browser](#)
- [Der Stylemanager](#)
- [Eine kleine Beispielsitzung](#)
- [KDE-Terminal: konsole](#)
- [KDE-Programme](#)

KDE

KDE steht für K Desktop Environment und ist der erste Desktop, der für Linux entwickelt wurde. Bis dahin hat es diverse Fenstermanager für Linux gegeben, am erfolgreichsten war fvwm, den es in einer fvwm95-Version gab, der MS Windows 95 im Look and Feel sehr ähnlich war. Offensichtlich durch CDE inspiriert, sollte mit KDE darüber hinaus eine grafische Oberfläche gestaltet werden, die nicht nur eine Sammlung grafischer Programme ist, sondern bei denen die Programme miteinander kommunizieren und ineinander greifen.



Man kann sehen, dass das Panel, dessen Idee von CDE übernommen wurde, fast zur Menüleiste mutiert ist. KDE arbeitet mit einem umfangreichen Menübaum, der seine Wurzel im K-Button hat.

Gleichzeitig gibt es Icons zum Starten von Programmen oder Geräten. Beides ist an MS Windows 95 angelehnt. Im Panel sind ebenfalls Startbuttons untergebracht und Sie können den Umschalter zwischen den vier Bildschirmen sehen, die zu Anfang eins, zwei, drei und vier heißen. In diesem Fall ist die Fläche zwei zu Buch umbenannt. Rechts daneben befindet sich die Fensterliste, in denen alle Fenster angezeigt werden, ganz gleich, auf welchem Arbeitsbildschirm sie sich befinden. Es folgt der Button mit dem Schloss zum kurzfristigen Sperren des Arbeitsplatzes und der Exit-Button. Beides kennt man auch vom CDE her. Darauf folgt ein Schnellstarter. Er zeigt Symbole für typische Desktop-Applikationen, wie hier das Clipboard und den Organizer. Wie beim Macintosh kann das Panel durch einen Klick im rechten Rand versteckt werden. Eine andere Idee vom Macintosh wurde übernommen: ein Doppelklick auf den Verschiebepfeil eines Fensters läßt das Fenster verschwinden; nur der Pfeil bleibt stehen. Ein weiterer Doppelklick und das Fenster ist wieder da. KDE-typisch ist es, dass die Objekte zum Starten nur einfach und nicht doppelt angeklickt werden. Nach den ersten Irritationen am Anfang macht dies die Bedienung relativ schnell. Es führt aber auch hin und wieder dazu, dass etwas startet, das man nur aus Versehen erwischte. Wenn das zu sehr stört, kann es auch umkonfigurieren.

Das Panel

Das Panel von KDE sieht auf den ersten Blick wie eine Menüleiste aus. Es ist in großem Maße über ein Menü, das über die rechte Maustaste erreichbar ist, konfigurierbar. Alternativ können Sie auch das Kontrollzentrum von KDE verwenden. Auf dem Panel lassen sich Icons zum Starten von Programmen und beliebig viele Menüs ablegen. Beides läßt sich wiederum über die rechte Maustaste verschieben oder entfernen.

Das Panel enthält wie das von CDE eine Umschaltbox, die zwischen den virtuellen Bildschirmen, den Arbeitsflächen umschalten kann. Durch Anklicken eines Schalters im Arbeitsflächenumschalters können Sie die Arbeitsfläche wechseln. Auch die Namen, die zu Anfang einfach eins, zwei, drei und vier lauten, sind leicht zu anzupassen, indem Sie auf ein angewähltes Fenster noch einmal klicken. Sofort erscheint ein Textcursor mit dem sich der Text an Ort und Stelle korrigieren läßt. Die Anzeige kann entweder die Namen der Fenster zeigen, die Nummern, die natürlich kleiner sind und helfen, wenn nicht viel Platz ist oder als kleine Vorschau, die die Umrisse der Fenster auf den jeweiligen Arbeitsflächen zeigt. Fenster können zu jedem Zeitpunkt über ihr Fenstermenü auf eine andere Arbeitsfläche bewegt werden. Auf diese Weise können Sie auch mit einem Bildschirm zurecht kommen, der keine große Auflösung hat.

Neben dem Arbeitsflächenumschalter befindet sich eine Taskliste, in der jedes gestartete Programm seine Titelleiste ablegt. Gleichartige Programme teilen sich eine Leiste, die beim Anklicken die einzelnen Fenster auffächert. Mit dieser Hilfe können Sie schnell zwischen Programmen wechseln. Auch minimierte Fenster legen sich nicht auf den Desktop, sondern bleiben innerhalb dieser Fensterliste.

Rechts neben der Taskleiste liegt ein Button mit einem Schloss. Damit kann der Bildschirm kurzfristig gesperrt werden. Wird er angeklickt, wird der Bildschirm schwarz und es erscheint eine Eingabebox für das Passwort. In diesem Zustand kann sich nur derjenige anmelden, der den Bildschirm gesperrt hat. Darunter befindet sich Ausschalter. Das Symbol zeigt einen Kreis mit einem senkrechten Strich darin, der dem Ausschaltssymbol einiger Elektrogeräte nachempfunden wurde. Damit verlassen Sie den Desktop.

Es folgt ein Platz mit kleinen Symbolen, beispielsweise für den Mixer oder das Clipboard oder den Organizer. Das Panel wird durch eine Zeitanzeige abgeschlossen.

Ganz rechts ist ein unscheinbarer Button mit einem kleinen rechten Pfeil. Klicken Sie diesen an, verschwindet das gesamte Panel nach rechts bis auf einen kleinen Button mit einem kleinen linken Pfeil, über den Sie sich das Panel wieder holen können. Auf diese Weise können Sie den Gesamtbildschirm nutzen, wenn Sie das Panel nicht brauchen.

Icons

Da sich minimierte Fenster nicht wie bei den früheren Fenstermanager als Icon auf dem Desktop sammeln, sondern in der Fensterliste des Panels abgelegt werden, konnten die Entwickler von KDE ohne das Risiko einer Verwechslung, Icons als Startsymbole für Programme, Verzeichnisse oder Geräte verwenden. Bei KDE können Sie ein solches Icon direkt auf dem Hintergrund anlegen. Dazu klicken Sie mit der rechten Maustaste auf den Hintergrund. Es erscheint das Kontextmenü. In der ersten Zeile steht »Neu« und dahinter die erzeugbaren Elemente: Verzeichnis, Dateisystem, Programm, URL oder Mimetyp. Nach seiner Erzeugung muss das Icon konfiguriert werden. Dabei unterscheiden sich die Icons je nach Typ.

- [Verzeichnis] Es erscheint eine kleine Dialogbox, die lediglich nach dem Namen fragt. In diesem Verzeichnis können Desktop-Objekte geordnet werden. Wie alle Objekte des Desktops befindet sich auch das angelegte Verzeichnis im Verzeichnis Desktop, das sich wiederum im Heimatverzeichnis des Anwenders befindet.
- [Dateisystem] Nach der Frage des Namens erscheint das Icon und sofort darauf ein Eigenschaftendialog. Darin ist vor allem die letzte Schaltfläche »Gerät« wichtig. Hier wird das Device aus dem Verzeichnis /dev eingetragen und der Mountpoint, auf dem es per mount in den Verzeichnisbaum eingebunden werden soll.

Seit der Version 2 von KDE wird im Menü zwischen Festplatte, Diskette und CD-ROM unterschieden. Das Gerät wird über eine Liste ausgewählt, die sich aus den Einträgen der Datei /etc/fstab ergibt. Damit erkennt KDE auch automatisch das Verzeichnis, in das es eingebunden werden soll.

- [URL] Zunächst wird wieder der Name des Icons, dann auf der letzten Seite die tatsächliche URL festgelegt.
- [Anwendung] Unter Ausführen wird der Aufrufname des Programms eingetragen. Steht das Programm im PATH, so braucht der Pfadname nicht angegeben werden.

Auch Iconbeschreibungen werden in Textdateien abgelegt. Sie finden sich im Heimatverzeichnis im Unterverzeichnis Desktop. Ein Icon für eine Anwendung kann unter der Haube beispielsweise so aussehen: Die Datei ist der besseren Übersicht halber um die diversen Übersetzungen des Namens gekürzt.

```
# KDE Config File
[KDE Desktop Entry]
Name[fi]=Sovellus
SwallowExec=/home/t345-1.33/t345
SwallowTitle=
Name[ru]=ððÉïïÖÄîÉÅ
BinaryPattern=/home/t345-1.33/t345;
Name=Application
Name[da]=Anvendelse
MimeType=
Name[de]=Acer T50 Konfiguration
Exec=acert50
Icon=PhoneTT.xpm
TerminalOptions=
Path=
Type=Application
Name[es]=Aplicaciones
Comment[de]=
Terminal=0
```

Man kann sehen, dass in den Iconbeschreibungen bereits die Beschriftung in verschiedenen Landessprachen hinterlegt sind. Während das Icon auf einem deutschen System »Acer T50 Konfiguration« heißt, würde dort in einer dänischen Umgebung »Anvendelse« stehen. Letzteres heißt einfach Anwendung und zeigt, dass das Icon wohl noch nicht in alle Sprachen übersetzt wurde.

Konqueror, Dateimanager und Browser

Der Dateimanager des KDE heißt Konqueror. Die erste Auffälligkeit ist, dass man die Objekte, die er zeigt, nur einmal anklicken muss, um sie zu starten. Dies ermöglicht eine ungewohnte, aber schnelle Navigation durch den Verzeichnisbaum. Sie können durch jeweils einen Klick auch durch die Verzeichnisse navigieren. Alternativ können Sie die Position auch durch Eingabe des Pfades in die Adressleiste ändern. Daneben lassen sich auch URLs eingeben und der Konqueror dient sogar als Browser. Sie können ihn von der Konsole mit dem Namen `konqueror` starten.



Konqueror hat eine Werkzeugleiste mit den üblichen Schalter eines Browsers, die sich aber auch gut zum Navigieren durch das Dateisystem verwenden lassen. Auffallend ist der Pfeil nach oben, der den üblichen Links- und Rechtspfeil ergänzt. Dieser wechselt in das Elternverzeichnis, während der Linkspfeil das zuvor angewählte Verzeichnis anzeigt. Dieser feine Unterschied kann sich als sehr praktisch erweisen.

Die Verzeichnisansicht in der linken Fensterhälfte erinnert sehr an den Explorer von MS Windows. In der Trennleiste zur Dateianzeige sind einige Schalter, die es ermöglichen, den Verzeichnisbaum an der Wurzel oder im Heimatverzeichnis beginnen zu lassen. Auch Webseiten und Lesezeichen lassen sich hier anzeigen. Mit dem Schließschalter lässt sich der Navigationsbereich abschalten. Mit der Funktionstaste F9 können Sie ihn wieder erscheinen lassen.

Der wichtigste Bereich ist für einen Dateimanager natürlich das Anzeigen von Dateien. Die Dateien können als Symbol, mehrspaltig, detailliert und in Baumansicht angezeigt werden. Die Symbolansicht und die mehrspaltige unterscheiden sich in erster Linie in der Größe der Symbole. Die Symbole zeigen den Dateityp an, die Konqueror in erster Linie anhand der Dateiendung erkennt. Den Nachteil, dass in dieser Darstellung nicht mehr Informationen als den Dateinamen und den Dateityp erfährt, gleicht

Konqueror dadurch aus, dass er weitere Informationen anzeigt, wenn Sie die Maus über dem Symbol einen Moment stehen lassen. Dann erscheinen Informationen wie Größe und Rechte, aber auch typspezifische Informationen wie die Länge in Sekunden bei Musikstücken. Bilder bekommen als Symbol eine Verkleinerung ihres Inhalts zugeordnet.

Detailansicht und Baumansicht haben nur sehr kleine Symbole, dafür zeigen sie spaltenweise Name, Größe, Dateityp und andere Informationen der Dateien an. Durch das Klicken auf die Spaltenüberschrift lassen sich die Dateien nach den verschiedenen Kriterien sortieren. Die Baumansicht setzt links neben die Dateinamen eine Hierarchie, mit der Sie durch die Verzeichnisse navigieren können. Anwendern, die vom Macintosh kommen, dürfte diese Ansicht bekannt sein.

Der Konqueror erkennt den Typ einer Datei an der Endung des Namens. Intern führt er eine Tabelle, in der zu jedem Dateityp eine Liste von Anwendungen zugeordnet ist, die diesen Dateityp verarbeitet. Sobald Sie eine solche Datei anklicken, startet Konqueror das entsprechende Programm. Darüber hinaus kann er aber auch Dateien erkennen, die in diesen Listen nicht vorhanden sind. In den Einstellungen des Konquerors finden Sie unter dem Titel »Dateizuordnungen« alle bekannten Dateierweiterungen. Sie können hier die Zuordnungen ändern und neue anlegen. Auch die Symbole der Dateien können Sie hier festlegen.

Sie kennen Lesezeichen vom Browser, um Adressen von Webseiten zu notieren. Lesezeichen funktionieren auch beim Konqueror in der gewohnten Art. Der Konqueror bietet die Möglichkeit, auch eigene Verzeichnisse auf der Platte mit Lesezeichen zu versehen. Für den Browserbetrieb ist es praktisch, dass man über das Lesezeichenmenü auch die Lesezeichen des Netscape Navigators verwenden kann.

Ein sehr praktisches Feature ist es, dass Sie mit einem kurzen ctrl-T ein Terminal erzeugen können, das sich im gleichen Verzeichnis befindet wie die Dateianzeige.

Der Konqueror kann nicht nur lokale HTML-Dateien korrekt darstellen, er kann auch als vollständiger Browser arbeiten. Dabei beherrscht er alle wichtigen Technologien wie Java, JavaScript, CSS und kann auch mit Plugins umgehen.

Der Stylemanager

Das KDE Kontrollzentrum ist eine Zusammenstellung der diversen Einstellungsdialoge des Dateimanagers, der Bildschirmanzeige und anderer grundlegender Programme. Zur Anpassung des Startmenüs dient das Programm `kmenuedit`. Mit diesem Programm werden die von den Icons bekannten `kdeLink`-Dateien in den Menübaum eingearbeitet.

Eine kleine Beispielsitzung

Die Aufgabenstellung für die Beispielsitzung soll es sein, sich anzumelden, ein Terminalfenster zu öffnen und zu schließen, sich auszuloggen und zu guter Letzt den Rechner geregelt herunterzufahren.

Nach dem Einschalten eines mit KDE installierten Rechners erscheint als Eingabeforderung eine einzelne Dialogbox, in der unter anderem ein Feld für die Benutzerkennung und eines für das Passwort existiert. Nach Eingabe der Benutzerkennung, die durch Return abgeschlossen wird, wechselt die Eingabeaufforderung zum Passwort, das blind einzugeben ist. Es erscheint als Rückmeldung lediglich ein Stern für jeden getippten Buchstaben.

Nun bieten sich drei Arten, an ein Terminalfenster zu gelangen. Am einfachsten ist es, auf dem Panel einmal auf das Symbol zu klicken, das so aussieht wie ein Monitor mit einer kleinen Muschel davor.



Alternativ verwenden Sie das KDE-Menü. Sie finden den Ausgangsbutton links unten mit einem großen K darauf. Dort wählen Sie nacheinander die Untermenüpunkte System - Shell.

Schließlich gibt es die Möglichkeit, über Alt-F2 eine Befehlszeile zu aktivieren und dort einfach `xterm` oder `konsole` einzugeben.

In allen Fällen erscheint ein Terminalfenster auf dem Sie arbeiten können, wie von einer textuellen Eingabe. Durch `ctrl-D` oder den Befehl `exit` schließen Sie das Fenster wieder.

KDE verlassen Sie wieder, indem Sie wiederum den linken unteren Button mit dem großen K anklicken und den Punkt »Abmelden« wählen. Es erscheint noch einmal eine Rückfrage, die Sie bestätigen und schon sind Sie wieder in der Anmeldemaske.

Im Anmeldedialog findet sich ein Button mit der Beschriftung »Beenden«. Klicken Sie ihn an, erscheint eine Dialogbox, die anbietet, den Rechner herunterzufahren, neu zu booten oder abzurechnen. Nachdem Sie »Herunterfahren« gewählt haben, können Sie die Meldungen des Rechners beim Shutdown beobachten.

KDE-Terminal: konsole

Das Programm `konsole` ersetzt `xterm` auf der KDE-Oberfläche. Bedauerlicherweise wurden einige der Standards bei den X Optionen ignoriert und durch eigene, nicht wirklich eingängigere Varianten ersetzt.

[Optionen von `xterm` und `konsole`]
C|C|L KDE & `xterm` & Bedeutung

- caption & -title & Im Titelfeld erscheinender Text
- vt_sz & -geometry & Größe des Fensters
- display & -display & Starten auf einem anderen X-Server
- e & -e & Start eines Kommandos
- ls & -ls & starte als Login Shell

Ansonsten ist es eine gelungene, moderne Umsetzung des `xterm`. Wer die Standard X-Kommandos nicht missen will, kann aber auch unter KDE das `xterm` in gewohnter Weise starten. Dies ist beispielsweise dann erforderlich, wenn Sie die Möglichkeiten von verschiedenen Hintergrundfarben nutzen wollen.

Eine hilfreiche Option ist `-workdir`, mit der Sie die Konsole gleich in ein bestimmtes Arbeitsverzeichnis

setzen können.

KDE-Programme

Im Lieferumfang des Desktops KDE gibt es viele Programme, die teilweise kleine, nützliche Helfer sind, aber auch ausgewachsene Applikationen. Hier sollen nur einige kurz vorgestellt werden, um eine Vorstellung von der Vielfalt und der Leistungsfähigkeit zu bekommen. Die Reihenfolge ist zufällig.

- [kpager]
Erzeugt einen Überblick über alle Desktops und zeigt in jedem den Hintergrund und die Fenster verkleinert an. Durch Anklicken der Felder wird zwischen den Desktops umgeschaltet.
- [ark]
Archivverwalter. Hiermit werden Inhalte von tar, tgz und zip Dateien angezeigt und verwaltet.
- [kfloppy]
Damit können Floppies sowohl im MS-DOS als auch im ext2-Format formatiert werden.
- [kpm]
Grafischer Ersatz für top. Die Bedienung ist allerdings eingängiger und Sie können durch Anklicken der entsprechenden Spalte nach verschiedenen Kriterien sortieren.
- [kwrite]
Ein Editor mit Syntaxhighlighting für diverse Sprachen (C++, Java, HTML, Latex etc.).
- [kiconedit]
Programm zur Erzeugung von Icons
- [kmail]
kma i l ist ein recht umfangreiches Mailprogramm mit einem Aufbau, der sich bei modernen E-Mailprogrammen mehr und mehr durchsetzt. In der linken Spalten stehen die Ordner, rechts oben eine Liste der Mails und darunter die Voranschau der gewählten E-Mail. Das Programm glänzt nicht nur mit einer großen Vielfalt an Optionen. Es ist auch in der Lage, von den unterschiedlichsten Mailprogrammen die Daten zu übernehmen. Da es selbst den Standards von UNIX in der Ablage seiner Daten folgt, kann man gut austesten, ob man mit dem Programm zurecht kommt.
- [kghostview]
Das Programm zeigt PostScript- und PDF-Dokumente an.
- [KonCD]
Dies ist eine Software zum Brennen von CDs. Um genau zu sein, ist es eigentlich nur die grafische Hülle. Denn KonCD verwendet die Programme cdrecord und mk i sofs, die im Anwendungsteil bereits beschrieben wurden (siehe S. cdrecord).

Unterabschnitte

- Das Panel
- Icons
- Dateimanager Nautilus
- Look and Feel
- Eine kleine Beispielsitzung
- Hinter den Kulissen
- GNOME-Programme

GNOME

GNOME (GNU Network Object Model Environment) ist eine weitere Implementierung für einen Linux Desktop, die daraus entstand, dass KDE die kommerzielle Qt-Bibliothek benutzt und damit nach Ansicht der Autoren von GNOME die Idee der freien Software untergräbt. GNOME basiert auf freien Bibliotheken, in erster Linie auf der gtk+ (GIMP Toolkit), die bei der Erstellung des Grafikprogramms gimp entstand.



Das Bild zeigt einen GNOME Bildschirm. Er wird eingerahmt von zwei Panels, die sich auf den ersten Blick kaum ähneln. Unten in der Mitte dient das Panel als Buttonleiste für das GNOME-Menü und einige Anwenderprogramme. Das obere Panel enthält einen Arbeitsflächenumschalter und eine

Fensterliste aller gestarteten Applikationen. Dieses Panel erscheint nur, wenn Sie mit dem Mauszeiger an die obere Kante des Bildschirms fahren und verschwindet wieder, wenn Sie dessen Arbeitsbereich mit der Maus wieder verlassen.

Rechts oben im Vordergrund sehen Sie Nautilus, den Dateimanager des GNOME Desktops. Hier zeigt er die Dateien des Verzeichnisses namens »linux« in Symboldarstellung an. Das hintere Fenster links unten zeigt den Organizer Evolution.

Das Panel

Auch GNOME hat wie KDE ein Panel, das einen oder mehrere Menübäume enthält und Schnellstartflächen für Applikationen aufnehmen kann. GNOME kann sogar mehrere Panels gleichzeitig anlegen, die rund um den Bildschirm angeordnet aber auch frei manövrierbar sein können. Dazu kommt, dass GNOME unterschiedliche Arten von Panels kennt. Die Kantenpanel können an jeder Seite des Bildschirms angeordnet werden. Sie können Sie jederzeit mit der mittleren Maustaste verschieben.

Recht witzig ist eine besondere Variante, die Menüpanel genannt wird und am oberen Rand positioniert wird. Einem Macintoshbenutzer dürfte alles sehr bekannt vorkommen. Sogar die Taskliste unter dem Menüpunkt rechts oben arbeitet wie bei MacOS.

Wenn Sie mit der rechten Maustaste auf den Panelhintergrund klicken, erhalten Sie ein Menü, in dem Sie einen Punkt namens Panel finden. Hier können Sie die Eigenschaften des Panels beeinflussen, aber vor allem können Sie hier Objekte hinzufügen. Die möglichen Objekte sind:

- [Menü]
Sie können hier ein GNOME Hauptmenü erzeugen.
- [Starter]
Sie können ein Icon auf dem Panel erzeugen, über das Sie durch einfaches Anklicken eine Applikation starten können. Vom Programmnamen über das Icon, den Befehl und dessen Parameter ist der Starter weitgehend konfigurierbar. Sie können sich einen Starter auch aus dem Menü auswählen.
- [Applet]
Applets sind kleine Programme, die kein eigenes Fenster benötigen, sondern dafür programmiert wurden, im Panel zu laufen. Sie bekommen bei Auswahl dieses Punktes eine große Auswahl solcher Programme angeboten. Typische Anwendungen sind Uhren, Statusanzeigen, diverse Umschalter, Fensterlisten und Arbeitsflächenumschalter.
- [Schublade]
In einer Schublade können weitere Starter abgelegt werden. Auf diese Weise können Sie recht viele Starter verwenden, ohne dass das Panel überquillt.
- [Sonderobjekte]
Dazu zählen vor allem der Button zum Ausloggen und der zum Sperren des Bildschirms.

Auch im GNOME Panel ist ein Umschalter für mehrere Arbeitsschirme eingebaut. Allerdings ist das bei GNOME ein großer, virtueller Arbeitsbereich von vier Fenstern. Sie können diese sogar so einstellen, dass man durch Berührung der Grenze mit der Maus hinüberwechselt. In jedem Fall können Sie Fenster auch über die Grenzen hinübertreten lassen. Hinzu kommt die Möglichkeit mehrere Arbeitsbereiche zu definieren, die den Arbeitsflächen von KDE entsprechen. Diese haben dann jeder vier Arbeitsschirme der oben genannten Art. Unten sehen Sie zwei Arbeitsflächen mit je vier Schirmen im Arbeitsflächenumschalter.



Icons

Icons lassen sich auf dem GNOME-Desktop wie bei KDE erzeugen. Sie klicken mit der rechten Maustaste auf den Bildschirmhintergrund. Es erscheint ein Menü. Hier sind die Punkte »Neues Verzeichnis« und »Neuer Starter« wichtig. Nach dem Erstellen eines Verzeichnisses wird ein Ordnersymbol auf dem Desktop erzeugt und ein entsprechendes Verzeichnis im Verzeichnis `~/gnome-desktop` angelegt.

Der Starter dient zum Ablegen von Icons, die doppelt angeklickt zum Starten von Programmen führen. Dazu wird der Name der Applikation ggf. mit Optionen benötigt und ein Name, unter dem das Symbol auf dem Desktop erscheint.

Ebenfalls über das Menü der rechten Maustaste können Icons für Wechselmedien oder MS Windows-Partitionen auf den Desktop gelegt werden. In dem Augenblick werden die Geräte automatisch in den in der Datei `/etc/fstab` (siehe S. `fstab`) angegebenen Verzeichnis eingehängt. Diese Symbole verschwinden von selbst, wenn die entsprechenden Geräte aus dem Verzeichnisbaum ausgehängt werden. Das kann einmal durch den Befehl `umount` erfolgen oder durch den Menüpunkt »Aushängen«, den Sie über die rechte Maustaste erreichen.

Dateimanager Nautilus

Nautilus ist der neuere Dateimanager des GNOME. Er ist komfortabler, aber auch ressourcenhungriger als sein Vorgänger gmc (GNOME Midnight Commander). Wie bei KDE wird auch bei GNOME der Dateimanager Nautilus als Verwaltungsprogramm für den Bildschirmhintergrund eingesetzt und Nautilus kann auch als Browser verwendet werden. Allerdings sind seine Fähigkeiten auf diesem Gebiet noch ausbaufähig.

Sie erhalten von Nautilus bereits vor dem Öffnen der Dateien sehr viele Informationen. So steht unter jedem Dateinamen auch die Dateigröße, bei Verzeichnissen die Anzahl der darin liegenden Objekte. Die Dateisymbole zeigen bei Bildern so genannte Thumbnails, also Verkleinerungen der Grafiken auf Daumennagelgröße. Bei der genaueren Betrachtung wird man allerdings staunen, wie groß mancher Daumennagel wohl ist., mit dem Inhalt des Bildes. Diese Miniaturbilder speichert Nautilus in einem Verzeichnis `.thumbnails`, das er zu dem jeweiligen Verzeichnis anlegt. Bei erneutem Aufsuchen des Verzeichnisses muss Nautilus die Bilder nicht neu erzeugen. Positionieren Sie den Mauscursor über einer Musikdatei, wird diese abgespielt. Bei Textdateien wird der Anfang dieses Textes verkleinert wiedergegeben.

Die Größe des Symbols können Sie verändern. Dazu klicken Sie das Icon mit der rechten Maustaste an und finden in dem erscheinenden Menü den Punkt »Icon strecken«. Das Icon erhält dann vier hervorgehobene Eckpunkte, an denen Sie es sozusagen auseinanderziehen können. In Kombination mit der Abschaltung der automatischen Positionierung, die Nautilus Layout nennt, können Sie Verzeichnisse recht frei gestalten.

Sie können Nautilus in eine Listenansicht umschalten. Es erscheinen die Spalten Dateiname, Größe, Typ und das Modifikationsdatum. Durch Anklicken des Spaltenkopfs können Sie die Dateien der Liste nach dem jeweiligen Kriterium sortieren.

In der Seitenleiste links neben dem Hauptfeld können Sie sich eine Baumansicht, eine Chronik der bislang angewählten Verzeichnisse und Notizen zum aktuellen Verzeichnis anzeigen lassen. Diese Notizen erscheinen bei jedem Betreten des Verzeichnisses und können jederzeit geändert werden. Die Baumanzeige müssen Sie zuerst über das Menü aktivieren, das Sie über die rechte Maustaste erreichen.



In Abbildung ist Nautilus umkonfiguriert, so dass die Dateien in Listendarstellung zu sehen sind, und die Seitenleiste einen Verzeichnisbaum zeigt. Im Baum sind auch Dateien zu sehen. Sie können das aber auch so umkonfigurieren, dass er nur Verzeichnisse enthält.

Nautilus erlaubt die Erzeugung von Verknüpfungen, wie man das von MacOS und auch von MS Windows kennt. Dabei wird ein symbolischer Link generiert. Das entsprechende Objekt erhält einen kleinen Haken, um es von normalen Dateien zu unterscheiden. Das Vorgehen gleicht dem auf dem Macintosh. Sie selektieren eine Datei, wählen das Kommando »Verknüpfung anlesen« aus dem Menü und im gleichen Verzeichnis wird sie generiert. Diese können Sie anschließend an jede beliebige Stelle schieben. Sie können aber auch nach der Art von MS Windows ein Objekt mit der rechten Maustaste verschieben und erhalten beim Loslassen ein Menü, in dem Sie aussuchen können, ob das Objekt verschoben, kopiert oder eine Verknüpfung erstellt werden soll.

Look and Feel

Im Gegensatz zu KDE ist bei GNOME wie bei anderen Oberflächen der Doppelklick üblich, um Applikationen zu starten oder Verzeichnisse zu öffnen. Allerdings kann man auch unter GNOME diese Voreinstellung über die Eigenschaften des Nautilus ändern.

Die Schließbox für die Fenster ist bei GNOME links statt rechts. Das dort üblicherweise zu findende Fenstermenü erreichen Sie über die rechte Maustaste. Dieses Verhalten kann so umgeschaltet werden, dass die Schließbox auf der rechten Seite erscheint. Die entsprechende Option finden Sie im Kontrollcenter unter dem »Aussehen« des Fenstermanagers Sawfish. Dort werden die Titelbuttons zwischen MS Windows, einigen anderen Vorbildern oder Standard umgeschaltet.

Durch Doppelklick auf den Fensterbalken verschwindet das Fenster und nur der Balken bleibt stehen. Das Fenster erscheint wieder beim nächsten Doppelklick.

Sie können an Icons Etiketten heften. Im Eigenschaftendialog, den Sie über die rechte Maustaste erreichen, finden Sie eine Reihe von Etiketten, die Sie auswählen können.

Insgesamt kann man sagen, dass GNOME sich stärker am Macintosh als an MS Windows orientiert, aber keine Hemmungen hat, gute Ideen aus welcher Quelle auch immer zu übernehmen.

Eine kleine Beispielsitzung

In dieser Beispielsitzung wird gezeigt, wie Sie sich anmelden, ein Terminalfenster öffnen und schließen, sich ausloggen und zu guter Letzt den Rechner geregelt herunterfahren.

Nach dem Einschalten eines mit GNOME installierten Rechner erscheint als Eingabeforderung eine einzelne Zeile für den »Username«. Nach Eingabe der Benutzerkennung, die durch Return abgeschlossen wird, erscheint eine Eingabeaufforderung für das Passwort, das blind einzugeben ist. Es erscheint als Rückmeldung lediglich ein Stern für jeden getippten Buchstaben.

Nun bieten sich zwei Arten an, an ein Terminalfenster zu gelangen. Am einfachsten ist es, auf dem Panel einmal auf das Symbol zu klicken, das so aussieht, als habe jemand gegen einen Fernseher getreten.



Alternativ können Sie das GNOME-Menü verwenden. Sie finden den Ausgangsbutton links unten ebenfalls in Form eines Fußabdrucks. Dort wählen Sie nacheinander die Untermenüpunkte Programme - System - GNOME-Terminal.

In beiden Fällen erscheint ein Terminalfenster auf dem Sie mit der Kommandozeile arbeiten können. Durch ctrl-D oder den Befehl exit schließt das Fenster wieder.

GNOME verlassen Sie wieder, indem Sie wiederum den linken unteren Button mit dem großen Fuß anklicken und den Punkt »Abmelden« wählen. Es erscheint noch einmal eine Rückfrage, die Sie bestätigen und schon sind Sie wieder in der Anmeldemaske.

Dort finden Sie im Menü unter System einen Punkt Herunterfahren. Es wird noch einmal gefragt, ob Sie wirklich den Rechner herunterfahren wollen. Danach können Sie die Meldungen des Rechners beim Shutdown beobachten.

Hinter den Kulissen

Seit GNOME 1.4 ist Sawfish der Standard als Fenstermanager. Er gilt als schnell und erweiterungsfähig. Dennoch ist diese Festlegung nicht bindend. Sie können GNOME auch mit einem anderen Fenstermanager verwenden.

Das GNOME Session Management ermittelt beim Ende einer Sitzung, welche Applikationen laufen und versucht bei der nächsten Sitzung, die Programme wieder zu starten und in den gleichen Zustand zu bringen, wie sie waren.

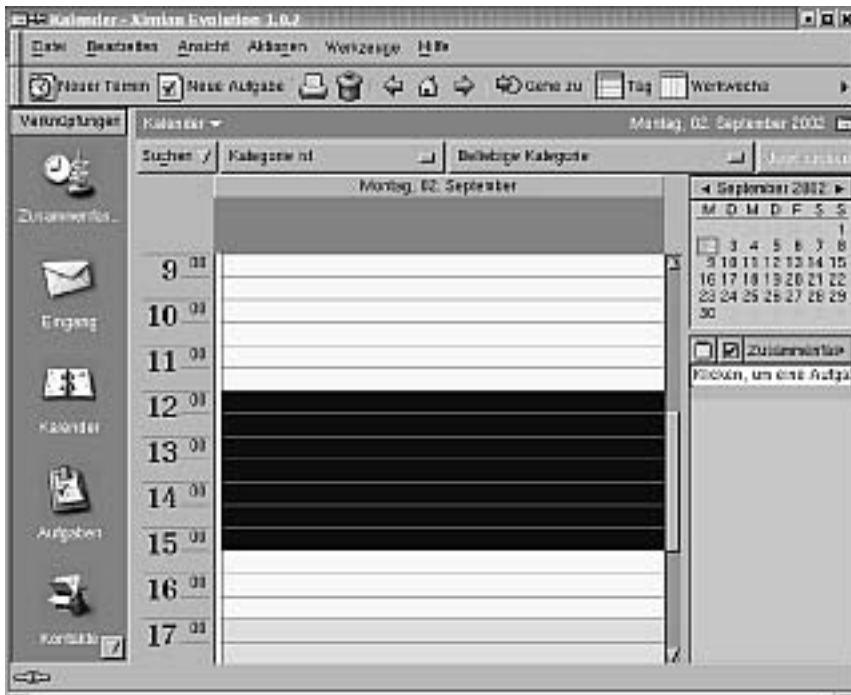
Für GNOME wurde das Komponentenmodell Bonobo entwickelt. Bonobo wird als ein sprach- und systemunabhängiger Schnittstellensatz zu CORBA beschrieben.vgl. die zu GNOME und Bonobo mitgelieferten README-Dateien. Damit ist GNOME unabhängig von bestimmten Sprachen oder Systemen und basiert auf anerkannten Standards, die weit über Linux hinaus neueste Technologie sind. Hinzu kommt, dass ein Komponentensystem als Basis einer grafischen Oberfläche sehr flexible Möglichkeiten bietet. Diese Architektur mag dafür verantwortlich sein, dass GNOME etwas träger als KDE zu reagieren scheint. Sie hat aber langfristig den Vorteil, dass Funktionalitäten eines Programmes ohne großen Zusatzaufwand als Modul zur Verfügung gestellt werden können. Dieses Modul kann dann von anderen Programmen verwendet werden, was zu einer schnelleren Entwicklungszeit und zu einem geringeren Speicherverbrauch führt.

GNOME-Programme

Wie unter KDE gibt es eine Flut von Anwendungen für den GNOME Desktop. Da man durchaus auch KDE-Programme unter GNOME ausführen kann und umgekehrt, haben Sie die Wahl beider Welten. Hier sollen ein paar Anwendungen aufgezählt werden.

- [Evolution]

Zu GNOME gehört das Paket Evolution, das bis ins Design verblüffende Ähnlichkeiten mit Outlook unter MS Windows hat. Es beinhaltet ein Mailprogramm, einen Kalender, eine Aufgaben- und eine Adressenverwaltung. Damit steht also GNOME ein so genannter Groupware Client zur Verfügung.



- [ggv]

Diese Version von Ghostview besticht durch viele hilfreiche Detaillösungen. Sie können damit sowohl PostScript- als auch PDF-Dateien betrachten.

- [abiword]

Eine Textverarbeitung, die zwar (noch) nicht so leistungstark wie die des StarOffice ist, aber durchaus kein Spielzeug ist.

- [gNumeric]

Diese Tabellenkalkulation ist leistungstark und komfortabel zu bedienen. Sie verfügt über zahllose Funktionen, hat aber zur Zeit noch keine eingebauten Diagramme.

- [gimp]

Das Grafikprogramm gimp ist eigentlich eher der Bibliothekslieferant von GNOME als ein Bestandteil des Desktop. Aber aufgrund der nahen Verwandtschaft kann und sollte man dieses Programm hier erwähnen.

Der Wettstreit der freien Desktops

Der Kritikpunkt an KDE bezüglich seiner kommerziellen Bibliothek qt ist weitgehend beigelegt, nachdem die Firma Troll Tech, die diese Bibliothek entwickelt hat, erhebliche Zugeständnisse an die KDE-Gemeinde gemacht hat. Die beiden Entwicklungen laufen derzeit nebeneinander her. Interessant an dem Wettbewerb zwischen den offenen Projekten ist, dass gute Ideen von der jeweils anderen Seite aufgegriffen werden. Die Panels haben große Ähnlichkeiten und der Konflikt ist offenkundig kein Überlebenskampf, wie dies im kommerziellen Bereich sehr schnell der Fall ist. Insofern profitieren derzeit die Anwender.

Sun liefert GNOME als alternative grafische Oberfläche für Solaris aus. Dabei kündigte Sun an, von CDE auf GNOME umsteigen zu wollen. vgl. http://www.sun.de/SunPR/Pressemitteilungen/2000/PM00_63.html Es ist zu vermuten, dass auf lange Sicht CDE an GNOME verlieren wird und es bleibt abzuwarten, inwieweit es eine Weiterentwicklung von CDE geben wird.

An dieser Stelle ist man leicht versucht, im Kaffeesatz zu lesen, welcher Desktop überleben wird. Ich habe derzeit auf meinem Laptop den einen Desktop und auf dem Arbeitsplatzrechner den anderen. Ich finde hier mal einen Nachteil und dort mal einen Vorteil. Derzeit ist die Entwicklung noch so stark in Bewegung, dass man endgültige Urteile zurückhalten sollte. Erfreulich ist, dass die Entwicklerteams von KDE und GNOME ein Miteinander der Programme ermöglichen.

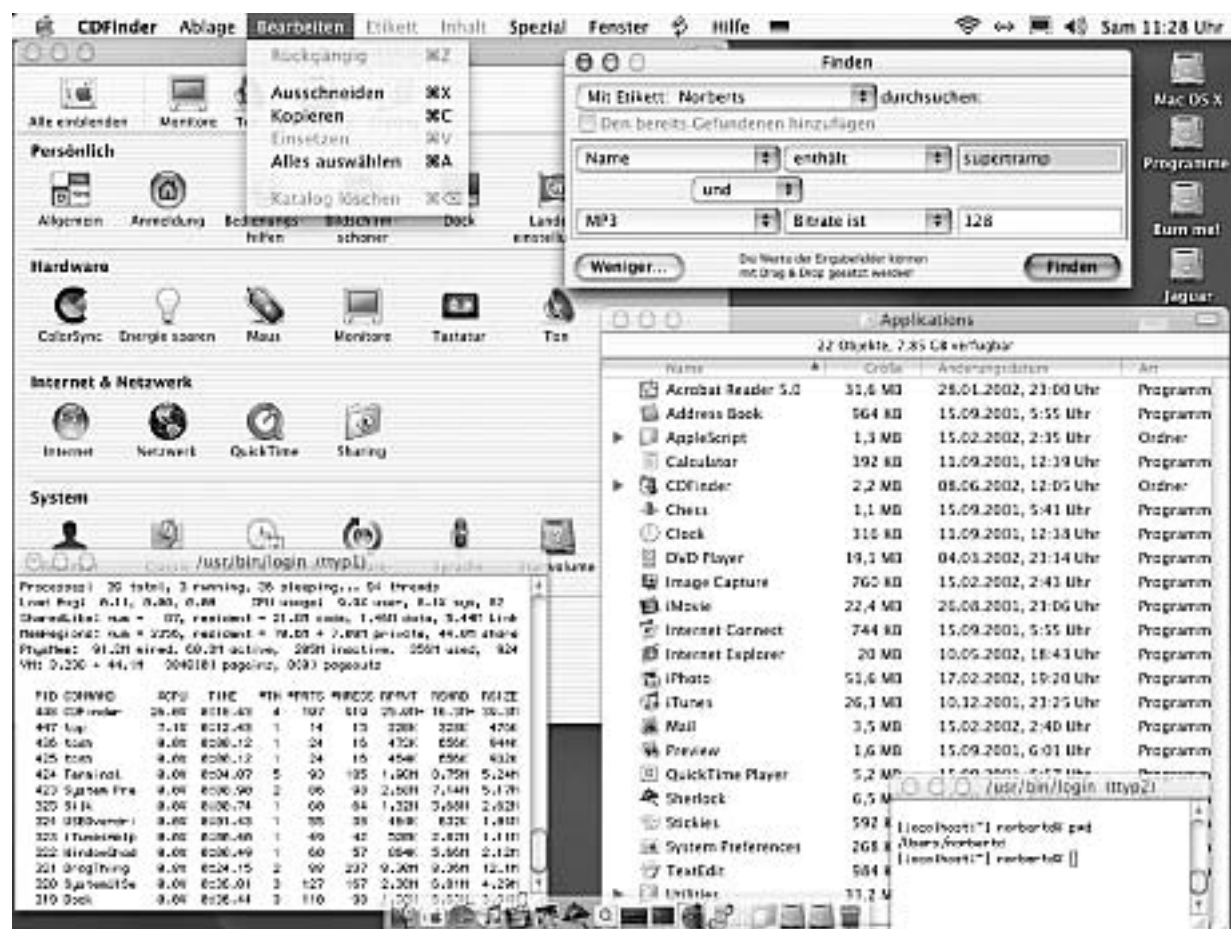
« GNOME | Desktops | MacOS X »

Unterabschnitte

- Eine kleine Beispielsitzung
- Programme für MacOS X

MacOS X

Völlig anders ist die Situation bei MacOS X. Das Betriebssystem des Macintosh hat zwar mit der neuen Version einen UNIX-Unterbau bekommen, aber deswegen wurde die grafische Oberfläche nicht auf der Basis des X Window System aufgesetzt. Stattdessen steht der Macintosh Desktop in der Tradition der klassischen Macintosh Oberfläche. Der Mac war immer schon ein Einzelplatzsystem, ganz ursprünglich war er nicht einmal ein Multitaskingsystem. Dafür war der Mac immer schon auf eine möglichst einfache und intuitive Bedienung ausgelegt. Dass die Firma Apple auf das X Window System umsteigen wird, ist wohl eher unwahrscheinlich. Auch wenn alles zum X Window System gesagte nicht für den Desktop von MacOS X gilt, ist es doch interessant, dass ein UNIX als robusten Unterbau für ein Einbenutzersystem eingesetzt wird.



Einen Macintosh erkennt man nach wie vor an der fest verankerten Menüleiste am oberen Bildschirmrand, dessen Menü von dem gerade im Vordergrund laufenden Programm bestimmt wird. Auf der Abbildung ist CDFinder <http://www.cdfinder.de> das aktive Programm. Dementsprechend sehen Sie dessen Menü in der oberen Leiste, leicht daran erkennbar, dass der Menüeintrag rechts neben

dem Apfel »CDFinder« heißt. Die Suchmaske von CDFinder, mit dem Sie nach Dateien in einem CD-Archiv suchen können, sieht man im Fenster rechts oben.

In dem großen Fenster links sind die Systemeinstellungen zu erkennen. Hiermit werden die Konfigurationsdialoge aufgerufen. Man erkennt die Untergliederung nach Themen. Vorn links und rechts unten ist jeweils ein Terminalfenster zu sehen. In der linken Sitzung läuft gerade das Programm top, in der rechten sieht man den Eingabeprompt.

Rechts unten sehen Sie den Dateimanager des MacOS X, den man beim Mac Finder nennt. Der Finder kann seine Dateien als Symbole oder zeilenweise, wie hier im Bild mit Minisymbol, anzeigen lassen. In dieser Darstellung sind auch weitere Eigenschaften der Dateien zu sehen. Oben rechts sehen Sie Laufwerke. Sie können den Finder so konfigurieren, dass er alle aktiven Dateisysteme auf dem Schreibtisch darstellen soll. Der Finder kann auch frisch eingelegte CD-Rohlinge brennen, wenn der Anwender das wünscht.

An dem aufgeklappten Menü ist schön zu sehen, dass auf dem neuen Desktop des Mac alles transparent ist.

Unten im Bild sieht man das »Dock« von Mac OS X. Es enthält links des Trennstriches eine Reihe von Programmsymbolen. Einige davon sind nach der Installation schon vorhanden, der Benutzer kann aber beliebig viele eigene hinzufügen. Außerdem tauchen alle vom Desktop gestarteten Programme dort auf. Sie erkennen sie an einem kleinen Dreieck unterhalb des Symbols. Der Teil des Docks rechts vom Trennstrich enthält Dokumente und den Papierkorb. Auch hier kann der Benutzer alles hineinziehen. Wenn man im Dock ein Symbol länger anklickt (also anklicken und festhalten), erscheint ein Menü mit einer Liste aller Fenster des Programmes, und einiger anderer Befehle (Beenden, Ausblenden, etc.) Programme können übrigens im laufenden Betrieb das im Dock angezeigte Symbol beeinflussen, um damit Statusinfos etc. anzuzeigen.

Da Mac OS X nicht auf X Window aufbaut, unterstützt es zur Zeit keine mehrfachen virtuellen Workspaces, also gibt es auch kein Umschalten zwischen ihnen. Immerhin kann das MacOS beliebig viele real vorhandener Bildschirme ansprechen und sie zu einem großen zusammenhängenden Desktop zusammenfassen.

Eine kleine Beispielsitzung

Die Aufgabenstellung der folgenden Beispielsitzung soll es sein, sich anzumelden, ein Terminalfenster zu öffnen und zu schließen, sich auszuloggen und zu guter Letzt den Rechner geregelt herunterzufahren.

MacOS X ist primär ein Einzelbenutzersystem. Insofern entfällt im Normalfall das Anmelden nach dem Einschalten des Rechners. Nur falls Ihr Rechner in ein Netzwerk eingebunden ist oder der Administrator das entsprechend konfiguriert hat, müssen Sie sich mit Benutzerkennwort und Passwort anmelden.

Ist das Terminal schon als Icon im Dock abgelegt, muß es einfach nur angeklickt werden. Ansonsten öffnen Sie mit Cmd-N ein neues Finder-Fenster, in dem entweder die eingehängten Volumes erscheinen, oder der Inhalt des Heimatverzeichnis des Benutzers. In beiden Fällen können Sie mit dem Menübefehl »Gehe zum Ordner« den Pfad »/Applications/Utilities« eingeben, und erhalten den Inhalt dieses Ordners angezeigt. Durch Doppelklick auf »Terminal« wird dieses gestartet.

Um sich vom System abzumelden, klicken Sie einfach im Apple-Menü, das ist das Menü unter dem blauen Apfel links oben, den Befehl »Abmelden« an und bestätigen die Rückfrage.

Programme für MacOS X

Mitgeliefert zum Mac OS X werden unter anderem folgende Programme:

- [iTunes]
Umfangreiche Software zum Erstellen, Verwalten und Abspielen von mp3 Dateien und anderen Sound-Files. iTunes spielt auch die Audio-CDs im Mac OS ab und dient dazu, portablen mp3-Playern wie dem iPod Daten zu vermitteln. Das Programm kann auch Audio- und mp3-CDs selbst brennen.
- [iPhoto]
Software zum Katalogisieren und Verwalten von Bildern. Das Programm kann diese auch direkt von digitalen Fotokameras herunterladen.
- [Address Book]
Programm zum Verwalten von Adressen. Das Programm wird auch von Mail und anderen Komponenten benutzt, so dass man seine Adressen nicht mehrfach eintragen muss.
- [iMovie]
Umfassendes und sehr einfach zu bedienendes Programm zum Schneiden von Filmen. Das Programm lädt die Daten auch gleich aus der digitalen Videokamera, und schickt sie nach der Bearbeitung wieder zurück. Die Software unterstützt aber nur FireWire-Kameras.
- [TextEdit]
TextEdit ist ein kleines Textprogramm, das aber leider alle Texte im Rich Text Format (RTF) speichert. Damit ist es aber leider zum Bearbeiten der typischen UNIX-Dateien unbrauchbar.
- [Utilities]
Enthält weitere Dienstprogramme, wie Konsole, Terminal, Disk Copy (zum Anlegen, Mounten und Brennen von Disk Images aller Art), Disk Utility (grafischer Ersatz von fsck und newfs, inklusive der Möglichkeit, Platten zu partitionieren), Process Viewer, Print Center (zum Verwalten von Druckerwarteschlangen) und den NetInfo Manager zur Verwaltung von Benutzern netzwerkweit.

« [Der Wettstreit der freien](#) | [Desktops](#) | [Das X Window System](#) »

Das X Window System im Netz

Der Bereich, wo X allen anderen grafischen Oberflächen überlegen ist, ist die Netzwerkfähigkeit. Während anderen Systemen erst mühselig beigebracht werden muss, wie Grafikschrime auch per Fernwartung bedient werden können, wurde das dem X Window System bereits in die Wiege gelegt. Ein X-Client spricht mit seinem X-Server über das X-Protokoll. Letzteres kann über TCP/IP transportiert werden und schon bietet sich die Möglichkeit, grafische Programme auf einer anderen Maschine zu bedienen.

Da die Nomenklatur im Anfang etwas irritierend ist, sei sie hier noch einmal klargestellt. Das X-Terminal ist der X-Server. Es bietet den Dienst von Grafikschrime, Tastatur und Maus dem anfragenden Programm an. Dementsprechend ist der X-Client die Maschine oder genauer der Prozess, der eine grafische Umgebung braucht.

Unterabschnitte

- [X-Programme über Netz starten](#)
- [X-Server Software in Betrieb nehmen](#)
- [Grafisches Einloggen übers Netz](#)
 - [X-Server sucht xdm](#)
 - [xdm sucht X-Server](#)
 - [Problemfälle](#)
 - [Linux als X-Terminal](#)
- [Thin Client](#)

X-Programme über Netz starten

Mit dem X Window System ist es sehr einfach, eine Anwendung zu starten und Bildschirm, Tastatur und Maus auf einen anderen Rechner umzulenken. Der X-Server, der Kontrolle und Ausgabe übernehmen soll, wird mit der Umgebungsvariable DISPLAY festgelegt. Die Variable muss exportiert werden, bevor der X-Client aufgerufen wird. Alternativ kann die Adresse des X-Servers statt in der Umgebungsvariablen DISPLAY auch direkt an den X-Client mit der Option -display übergeben werden.

Der X-Server wird durch den Hostname, ein Doppelpunkt, die Nummer des Displays, einen Punkt und den Bildschirm (engl. screen) angegeben.

Hostname:Display.Screen.

Der Hostname ist der übliche Name des Rechners, wie er beispielsweise in der Datei /etc/hosts hinterlegt wird. Der Display ist eine Einheit aus Bildschirm, Maus und Tastatur und wird je Rechner durchnummeriert. Da die wenigsten Rechner heutzutage noch mehr als einen solchen Arbeitsplatz zur Verfügung stellen, ist diese Zahl meistens 0. Mit dem Screen wird der Bildschirm des Displays bezeichnet. Wenn ein Arbeitsplatz mit zwei Bildschirmen ausgestattet ist, werden diese ebenfalls durchnummeriert. Auch dieser Fall ist eher selten und darum ist der Screen meistens 0 und kann auch inklusive Punkt völlig weggelassen werden. Soll die Anwendung auf dem lokalen Host ablaufen, kann der Name weggelassen werden, aber der Doppelpunkt und die folgende 0 muss bleiben.

Beispiel: gaston:0 oder localhost:0.0 oder :0

Wie fast überall braucht auch hier der Client das Recht, den Dienst des Servers in Anspruch zu nehmen. Sie erteilen einem X-Client dieses Recht durch den Befehl xhost.

[Freigabe des X-Servers: xhost]L|L Befehl & Bedeutung
xhost & Anzeige des derzeitigen Rechtstatus
xhost + & Freigabe des X-Server für alle fremden Rechner
xhost +Rechner & Freigabe für Rechner
xhost - & Sperren des X-Servers für alle fremden Rechner
xhost -Rechner & Sperren für Rechner

Als Beispiel wird das Programm xcalc, der X-Taschenrechner, auf dem Rechner asterix gestartet. Die Bedienung des Programms soll aber auf dem Rechner gaston erfolgen. Zunächst müssen Sie auf gaston zulassen, dass andere Rechner seinen Display benutzen.

```
gaston> xhost +  
access control disabled, clients can connect from any host  
gaston>
```

Danach wird eine Sitzung auf asterix gestartet. Das kann per telnet oder per Konsole erfolgen. Dort

wird `xcalc` aufgerufen mit der Option, `gaston:0` als Display zu verwenden. Der Aufruf würde folgendermaßen aussehen, wenn Sie die Option `-display` benutzen.

```
asterix> xcalc -display gaston:0 &  
[1] 3289  
asterix>
```

Bei Programmen, die die Option `-display` nicht akzeptieren, funktioniert normalerweise dennoch die Variante mit der Umgebungsvariable `DISPLAY`. Hier wird die Variable `DISPLAY` auf `gaston:0` gesetzt und dann per `export` zur Umgebungsvariablen gemacht.

```
asterix> DISPLAY=gaston:0  
asterix> export DISPLAY  
asterix> xcalc &  
[2] 3291  
asterix>
```

Diese Möglichkeiten sind darum so praktisch, weil sie es erlauben, mit einem Bildschirm auf mehreren Maschinen zu arbeiten und es sogar ermöglichen, auf Maschinen grafische Programme starten, auch wenn diese selbst gar keine Grafikhardware besitzen.

Eine ganz typische Anwendung ist das Administrationstool `sam` für das HP-UX. Dieses kann auf Terminals oder als X-Anwendung laufen. Wer aber an seinem HP-Server keinen X-Server hat, kann leicht über die Umgebungsvariable `DISPLAY` die Maschine von einem Linuxrechner mit X administrieren. In solchen Fällen ist das vielleicht die einzige X-Anwendung, die je von dieser Maschine gestartet wird.

« [Das X Window System](#) | [Das X Window System](#) | [X-Server Software in Betrieb](#) »

X-Server Software in Betrieb nehmen

In diesem Abschnitt soll als Bildschirm eine X-Server Software verwendet werden, die auf der Basis eines Fremdsystems wie MS Windows oder MacOS läuft. Der X-Client kann über TCP/IP auch zu dieser Software Kontakt aufnehmen und sie arbeitet als grafisches Display.

Als Beispiel wird ein Macintosh mit der X-Server Software MI/X von der Firma MicroImages verwendet. Das Programm kann kostenlos aus dem Internet heruntergeladen werden. Die Firma bietet auch einen X-Server für MS Windows an, der allerdings kostenpflichtig ist.

Quelle: <http://www.microimages.com>

Nachdem der X-Server gestartet wird, ist der Bildschirm typischerweise leer. Auf dem X-Client, der im Beispiel ein Linuxrechner mit KDE ist, muss das Programm gestartet werden. Der X-Server läuft auf dem Rechner namens mac. Dazu starten Sie auf gaston das Programm xterm.

```
xterm -display mac:0 &
```

Nach kurzer Zeit sollte sich auf dem Mac ein Fenster mit einem xterm melden. Dabei hat das xterm noch keinen Rahmen. Das ist nicht verwunderlich, da die Rahmen von X-Programmen vom Fenstermanager erzeugt werden. Auf dem nackten X-Server läuft kein Fenstermanager. Allerdings bietet MI/X an, lokal twm zu benutzen. Um dem abzuhelpen, wird als nächstes ein Fenstermanager gestartet. Im xterm wird kwm, der Fenstermanager des KDE, gestartet. Anschließend erscheint ein Rahmen und ein Titelfeld um das X-Term. Danach können Sie durch das Starten von kfm den Desktop initialisieren. Es erscheinen die Icons. Zu guter Letzt können Sie durch den Aufruf von kpanel die untere Werkzeugleiste starten. Insgesamt gewinnt man fast den Eindruck, direkt an der Linux-Maschine zu sitzen. Hier noch einmal die drei Aufrufe:

```
kwm &  
kfm &  
kpanel &
```

Sie brauchen also ein einziges xterm auf dem X-Server, um alle möglichen X-Clients auf gaston zu starten. Dabei vererbt sich offenbar die Displayumgebung an alle Programme, die von der Sitzung gestartet werden.

[« X-Programme über Netz starten](#) | [Das X Window System](#) | [Grafisches Einloggen übers Netz](#) »

Unterabschnitte

- [X-Server sucht xdm](#)
 - [xdm sucht X-Server](#)
 - [Problemfälle](#)
 - [Linux als X-Terminal](#)
-

Grafisches Einloggen übers Netz

Die Netzwerkfähig des X Window Systems reicht noch weiter. Es bietet die Möglichkeit, dass Sie sich mit einem X-Server so ähnlich wie mit einem Terminal anmelden. Es ist also möglich, sich an ein und der selben Maschine mit mehreren X-Servern oder X-Terminals unabhängig voneinander zu arbeiten.

Dabei kennt das X auch noch zwei unterschiedliche Strategien.

In dem einen Ansatz sucht der X-Server nach X-Clients an denen er sich anmelden kann. Hier wird vom X-Client lediglich die Berechtigung zum Anmelden freigeschaltet. Die Initiative geht vom X-Server aus und der Zentralrechner reagiert auf Anfragen.

In der einen Variante konfigurieren Sie im Zentralrechner, welche X-Server er für eine Anmeldung zulassen soll. Vergleichbar mit dem `getty` für Terminals läuft dann jeweils für einen X-Server ein Dämonprozess, der auf dessen Anmeldung wartet. Die Initiative geht vom Zentralrechner aus und die X-Server sind passiv.

X-Server sucht xdm

Der zentrale Dämon für das grafische Einloggen über das Netzwerk ist `xdm`, der auch benutzt wird, um sich an einer lokalen Workstation grafisch anzumelden. Damit er auch Netzanfragen beantwortet, muss `xdm` aber erst konfiguriert werden. Die Konfigurationsdateien befinden sich im Verzeichnis `/etc/X11/xdm`.

Die Datei `xdm-config` ist, wie der Name schon sagt, die wichtigste Konfigurationsdatei des `xdm`. Hier wird zunächst freigeschaltet, dass `xdm` Anfragen über das Netz akzeptiert. Dazu muss die Variable `DisplayManager.requestPort` gesetzt sein. Steht diese auf 0, wird ein Zugriff von außen verhindert. Der Standardport für den `xdm` lautet 177, wie Sie in der Datei `/etc/services` feststellen können.

<code>DisplayManager.requestPort:</code>	<code>177</code>
--	------------------

Mit dieser Zeile in der Datei `xdm-config` lässt `xdm` Anfragen über das Netzwerk zu. Ein X-Server darf anfragen, ob er sich hier einloggen darf. Damit wird der X-Server zum Client des `xdm`.

Die Berechtigung wird in der Datei `Xaccess` festgelegt. Standardmäßig steht hier eine Freigabe für alle interessierten X-Server.

Die erste Zeile ermöglicht einem X-Server die Query, also die direkte Anfrage an xdm. Die zweite Zeile ist von Bedeutung, wenn der Zugriff über einen Broadcast erfolgt. Dann fragt das X-Terminal über das Netz nach allen Rechnern, an denen es sich anmelden könnte. Wollen Sie nur bestimmte X-Server zulassen, können Sie hier Einschränkungen definieren. Beispiele und weitere Hinweise finden Sie in der Manpage von xdm.

Ein X-Server kann sich jetzt anmelden. Dazu muss der X-Server in den »Query«-Modus gesetzt werden und der Zielrechner muss angegeben werden. Dann erscheint nach kurzer Wartezeit der Anmeldebildschirm.

xdm sucht X-Server

Anders herum haben Sie die Möglichkeit, von xdm aus das X-Terminal zu aktivieren. Dazu werden in der Datei Xservers die Displays festgelegt. Für jeden eingetragenen X-Server wird ein eigener xdm-Prozess gestartet. Dazu werden beispielsweise folgende drei Ziele in der Datei Xserver aufgeführt.

```
:0 local /usr/X11R6/bin/x :0 vt07
pm7500:0 foreign
silver:0 foreign
```

Der erste Eintrag ist der, der bei einer Standardinstallation für den lokalen X-Server vorhanden sein sollte. Das vt07 zeigt an, dass sich die X-Sitzung auf der virtuellen Konsole 7 befindet. Unter Linux lassen sich mit der Tastenkombination Strg-Alt-Fx zwischen virtuellen Konsolen umschalten. Die nächsten beiden Einträge stellen einen eigenen xdm für jedes der beiden X-Terminals pm7500 und silver ab. Es reicht, nach der Änderung der Datei dem Display Manager mit `kill -1` ein Signal SIGHUP zuzusenden. Bei der nächsten Prozessanzeige finden Sie drei xdm-Prozesse in der Prozessliste. Sobald der X-Server passiv im Netz steht, erscheint dort der Anmeldedialog.

Problemfälle

In einigen Red Hat Versionen soll der Fontserver xfs nicht auf das Netzwerk eingestellt sein. vgl. XDMCP-HOWTO Chapter 2.6. Dazu ist in dem entsprechenden rc-Skript xfs in `/etc/rc.d/init.d` zu prüfen, ob xfs als Option `-port 7100` ist, oder mit `-port -1` abgeschaltet wurde. Die richtige Zeile lautet so oder ähnlich:

```
daemon xfs -droppriv -daemon -port 7100
```

Von sich aus bedient kdm zwar sofort die in Xservers aufgeführten X"-Terminals mit einem eigenen kdm-Prozess, er lässt aber keine Anfragen von X"-Terminals zu, die sich einloggen wollen. Dazu muss die Reaktion auf Anfragen per XDMCP (X Display Manager Control Protocol) erst freigeschaltet werden. Dies geschieht in der Startdatei des kdm namens `kdmrc`. Diese findet sich im Verzeichnis `/opt/kde2/share/config/kdm`. Es kann auch sein, dass der Pfad unterhalb des Verzeichnisses `/etc` zu finden ist. In dieser Datei sind folgende Einträge zu kontrollieren:

```
[Xdmcp]
Enable=true
```

```
xaccess=/var/x11R6/lib/xdm/xaccess  
willing=/var/x11R6/lib/xdm/xwilling
```

Es reicht nicht, die Änderungen mit `kill -1` dem Prozess `kdm` zu signalisieren. Die `kdmrc` wird, wie das `rc` im Namen schon vermuten lässt, nur bei Start des `kdm` ausgewertet. Es ist also ein Neustart des `kdm` erforderlich. Die Idee, den `kdm` aus der X Sitzung heraus direkt abzuschießen, sollten Sie nur weiterverfolgen, wenn Sie alle Daten gesichert haben. Besser ist es, mit `init 2` in den Multiuserlevel herunterzufahren und mit `init 5` wieder durchzustarten.

Wie an anderer Stelle erwähnt, gehören reine X-Terminals heute zu den echten Raritäten. Meist wird ein MS Windows-PC oder ein Mac als X-Terminal verwendet, indem entsprechende Software gekauft wird. Obwohl der Begriff Terminal vermuten lässt, dass man nicht viele Ressourcen benötigt, wird für das Anmelden und Arbeiten an einem modernen Desktop via X-Server einiges an Speicher und CPU-Leistung benötigt.

Linux als X-Terminal

Jede UNIX-Workstation hat natürlich einen X-Server eingebaut, mit dem sie normalerweise mit den lokalen X-Clients kommuniziert. Dieser X-Server heißt einfach `X`. Um eine Workstation als reines X-Terminal zu betreiben, reicht es aus, `X` mit den Parametern aufzurufen, die den X-Server veranlassen, sich bei einem anderen Rechner mit dem XDM-Protokoll anzumelden.

Der X-Server muss gewöhnlich als `root` gestartet werden. Falls die Maschine automatisch in den `xdm`-Modus bootet, müssen Sie sie erst wieder in den Runlevel 2 bringen.

```
init 2
```

Sind Sie als `root` eingeloggt und der X-Server ist heruntergefahren, dann gibt es drei Möglichkeiten, um `xdm` auf `gaston` zu erreichen:

```
X -query gaston
```

Der erste Aufruf ist die direkte Anfrage an `xdm`, sich bei `gaston` einloggen zu wollen.

```
X -indirect gaston
```

Die zweite Variante fragt den Host `gaston`, welche Rechner er kennt, bei denen man sich im Netz einloggen kann. In den meisten Fällen wird er dem Anfrager eine Liste anbieten, in der nur er selbst aufgeführt ist. Diese Auswahlbox ist der Chooser.

```
X -broadcast
```

In der dritten Zeile wird kein Rechner genannt. Der X-Server streut ins Netz die Anfrage, ob es einen `xdm`-Server gibt und zeigt die Anmeldebox des Rechners, der am schnellsten antwortet.

Geht die Initiative von `gaston` aus, dann wird der X-Server wie oben gesehen in der Datei `Xservers` aufgeführt und es wird explizit für diesen X-Server ein `xdm` gestartet. Um darauf zuzugreifen, lautet der Aufruf des X-Servers:

« X-Server Software in Betrieb | **Das X Window System** | Thin Client »

Thin Client

Insbesondere die Anbindung eines X-Servers an den xdm zeigt, dass der X-Server selbst nicht mehr können muss, als eine Netzkommunikation aufzunehmen und ein grafisches Terminal zu steuern. Tatsächlich ist ein solches Gerät völlig wartungsfrei und würde in Fragen der »Total Cost of Ownership« übersetzt heißt es etwa die Gesamtkosten des Besitzes. Es fasst die Kosten zusammen, die nicht nur durch die Anschaffung entstehen. In erster Linie sind das Wartung und Administration. alle Rekorde brechen. Die Software auf den Servern muss dann einfach nur als X-Client geschrieben werden.

Wer also nach dem Ideal des Thin Client strebt, braucht nicht auf zukünftige Entwicklungen zu warten. Die Technologie dafür wurde bereits in den 80ern entwickelt und ist absolut ausgereift. Nun werden zwar X-Terminals nicht mehr gebaut, aber das ist nicht weiter tragisch. Auf der Basis von Linux lässt sich leicht aus Standard PCs ein System zusammensetzen, das per DHCP (siehe S. dhcp) alle Netzinformationen bekommt, anschließend nur den X-Server startet, um sich an den Servern im Netz einzuloggen. Fällt ein solcher Rechner einmal aus, wird die Masterplatte eins zu eins kopiert und ein neuer Rechner ins Netz gestellt. Da die Geräte keine Sonderanfertigung sind, wären sie vermutlich sogar billiger als jeder speziell angefertigte Thin Client.

Die Tatsache, dass diese Technologie noch nicht im großen Stil umgesetzt wird, zeigt, dass das Problem an einer ganz anderen Stelle als an der technischen Machbarkeit liegt. Es ist die Angst der Sekretärin, dass die neue Textverarbeitung ganz anders ist als die alte. Es ist die Angst des EDV-Chefs, dass er vielleicht alles neu lernen muss und es ist die Panik des Lieferanten, dass er mit einem Schlag vom gefragten Experten zum Dilletanten wird. Aus diesem Grund ist die »Zukunftsbranche« wie keine Andere von der Angst vor Veränderungen geprägt.

Es gibt aber tatsächlich auch einen sachlichen Grund, der gegen den Thin Client spricht. Der Aufteilung der Aufgaben zwischen Client und Server ist nicht optimal. Der Server beschäftigt sich mit Arbeiten wie Menüaufbauten, die nicht zentralisiert sein müssen. Die Netzbelastung von X ist zwar nicht so üppig, aber immer noch stärker, als wenn nur die nackten Anfragen einer Client-Server-Architektur über das Netz gingen. Die Ressourcen auf dem Client werden nicht genutzt. Man entlastet also den billigen Arbeitsplatzrechner, um die teuren Komponenten zu belasten: das Netz und den Server!

Anwendungssoftware für UNIX

Seit einigen Jahren gibt es bereits einige Anwendersoftware, die auf den verschiedenen Plattformen angeboten werden und die unter X laufen. Einige dieser Anwendungen entstammen der MS Windows-Welt wie StarOffice oder Acrobat Reader. Einige Programme haben ihre Wurzeln eigentlich in der X-Umgebung und sind, als die PCs immer leistungsfähiger wurden, auch auf MS Windows portiert worden, da dort einfach ein größerer Markt existiert.

FrameMaker

FrameMaker ist eines der ältesten DTP-Programme. FrameMaker wurde zunächst auf UNIX-Workstations eingesetzt, da es eine leistungsfähige Umgebung braucht. Die damals verfügbaren Personal Computer waren noch nicht in der Lage, die entsprechende Leistung zu liefern.

Maple

Maple ist ein Computeralgebrasystem, das vor allem im universitären Umfeld eingesetzt wurde. Es ermöglicht die Darstellung von Funktionen und deren Export in diverse Formate, unter anderem auch .

StarOffice

Das Office der Firma Star Division wurde inzwischen von Sun aufgekauft und wird für die Plattformen Linux, Solaris und MS Windows bis zur Version 5.2 kostenfrei vertrieben. Ab Version 6.0 ist nur noch die Solaris-Version kostenlos. Allerdings gibt es immer noch das abgespaltene Produkt OpenOffice, das nach wie vor kostenlos zur Verfügung steht. Gegenüber der StarOffice Version fehlen ihr Wörterbücher, für die Sun selbst Lizenzen zahlen muss. Das Paket ist recht üppig gefüllt. Es gibt eine Textverarbeitung, eine Tabellenkalkulation, eine Datenbank, eine Bildbearbeitung, eine Zeichenanwendung. Präsentation, E-Mail und Kalender sind ebenfalls dabei. Da Sie alle möglichen Formate im- und exportieren können, kommen Sie ohne einen MS Windows-PC neben der UNIX-Maschine aus. In Tests schneidet StarOffice oft besser als Microsoft Office ab.

Sun begründet die Strategie, die Software kostenlos abzugeben, damit, durch Support das Geld zu verdienen. Tatsächlich ist natürlich offensichtlich, dass Sun einen Angriff auf die marktbeherrschende Position Microsofts durchführt. Gibt es ein Office für Linux, wird der Schritt von MS Windows weg leichter. Kann auf allen Plattformen mit den gleichen Dokumenten gearbeitet werden, fügt sich die UNIX-Plattform problemlos in die Unternehmens-EDV. Dann entscheidet irgendwann nur noch Stabilität und Virensicherheit und das wäre das Ende der MS Windows Monokultur.

Netscape Navigator

Der erste kommerzielle Browser wird längst kostenlos abgegeben und ist zum Open Source Projekt geworden. Der mitgelieferte Messenger dient als E-Mail und News Client. Auch hier ist Microsoft für die kostenlose Weitergabe indirekt verantwortlich. Um den Markt des Internet zu beherrschen, wird auch der Internet Explorer kostenlos abgegeben.

Acrobat Reader

Der PDF-Reader der Firma Adobe namens Acrobat Reader wird ebenfalls kostenlos für die verschiedenen Plattformen verteilt. Auch wenn ghostview inzwischen auch PDF-Dateien liest, gibt es Dateien, die man doch besser mit dem Acrobat Reader lesen kann. Übrigens wird das Programm von der Konsole mit `acroread` aufgerufen.

Das Satzsystem

Donald Knuth schrieb das Programmpaket und stellte es zur freien Benutzung zur Verfügung. Ursprünglich war es entwickelt worden, um mathematische Ausdrücke ohne großen Aufwand druckreif zu setzen. Dann wurde es von Leslie Lamport um dahingehend erweitert, ohne genauere Kenntnisse des Satzgewerbes perfekte Dokumente zu erzeugen, die aussehen, als wären sie gesetzt worden. Da das Programm hervorragende Ergebnisse produziert, hat es sich im Bereich Publikationen bei allen durchgesetzt, die bereit sind, sich auf das Lernen der nicht allzu schwierigen Satzsprache einzulassen.

Neben der klassischen Möglichkeit, den Quelltext mit einem normalen Editor wie `vi` oder `emacs` zu schreiben, gibt es inzwischen grafische Editoren wie `Lyx`, der wie ein WYSIWYG-Editor arbeitet, aber Quelltexte von erzeugt.

Ein großer Vorteil ist die Portierbarkeit der Dokumente. Da alles reiner ASCII-Text ist, gibt es keine Probleme mit der Bearbeitung oder Generierung durch andere Anwendungsprogramme. Da auf beinahe jeder Betriebssystemplattform verfügbar ist, ist es egal, auf welcher Maschine der Text erfasst wird. Dazu kommt, dass einige Werkzeuge mitgeliefert werden, die es ermöglichen -Dokumente nach HTML, nach PostScript oder PDF umzuwandeln.

Da der Quelltext reiner ASCII-Text ist, können Anwendungsprogramme verwenden, um Serienbriefe zu generieren, ohne dass Sie gleich die Datenstruktur einer Anwendung nachahmen müssen. Programmierer, die nicht nur einfach Listen auf einem Nadeldrucker erzeugen sollen, sondern Dokumente, die edel aussehen, haben damit ein ideales Ausgabemedium.

Zu guter Letzt sehen mit gesetzte Dokumente einfach sauber gesetzt aus, da das Setzen selbst übernimmt und damit verhindert, dass Dilletanten, mit ihrer Textverarbeitung bewaffnet, dem professionellen Setzer das Wasser in die Augen treiben. Das ist auch der Grund, warum dieses Buch in geschrieben wurde.

Unterabschnitte

- [FrameMaker](#)
 - [Maple](#)
 - [StarOffice](#)
 - [Netscape Navigator](#)
 - [Acrobat Reader](#)
 - [Das Satzsystem](#)
-

Programmierung

- **Programmierung von Shellskripten**
 - Erstellen und Start eines Shellskripts
 - Variablen
 - Ablaufsteuerung
 - Ein- und Ausgaben aus dem Skript
 - Start und Umgebung von Skripten
- **Perl**
 - Interpreter und Skript
 - Variablen
 - Interaktiv
 - Ablaufsteuerung
 - Dateien
 - Perl und UNIX
 - Grafische Oberfläche: Tk
 - Informationsquellen
- **Programmierwerkzeuge**
 - C-Compiler
 - make
 - Debugger
 - Analysewerkzeuge
 - Versionsverwaltung
 - Diverse Programmierhelfer
- **UNIX-Systemaufrufe**
 - Die Funktion main
 - Fehlerbehandlung: errno
 - Dateizugriffe
 - Verzeichnisse
 - Prozesse
 - Signale
 - Pipe
 - Fehlerbehandlung mit syslog
 - Zeitfunktionen
 - Benutzer und Gruppen
 - Grundlagen der Dämonisierung
 - Client-Server Socketprogrammierung
 - Reguläre Ausdrücke
 - Weitere Programmierschnittstellen

Programmierung von Shellskripten

Programmieren ist die hohe Kunst der Computerbenutzung. Es ist die Faszination, kreativ zu gestalten. UNIX ermöglicht einen fließenden Einstieg über die Skriptsprachen.

Die Shell ist nicht nur einfach ein Befehlsempfänger, sondern auch eine durchaus leistungsfähige Programmiersprache. Gemischt mit der Kombinierbarkeit der UNIX-Kommandos entsteht die Möglichkeit, auch komplexere Abläufe zu programmieren. Insbesondere im Bereich der Systemverwaltung ist diese Sprache einer Programmiersprache wie C sogar überlegen.

Shellskripten sind unter UNIX unentbehrlich. Man setzt sie immer dann ein, wenn komplexere Abläufe durch einen einfachen Aufruf erledigt werden müssen. Dazu gehören Administrationsarbeiten, die zeitversetzt durch `cron` oder `at` gestartet werden. Aufgaben, die per `sudo` an Anwender weitergegeben werden sollen, können in Shellskripten spezifiziert werden. Als `rc`-Skripte, die beim Booten gestartet werden, sind sie bei der Konfiguration des Systems erforderlich. Wie wichtig Skriptsprachen sind, sieht man daran, dass auch die PC-Betriebssysteme wie MS Windows und Macintosh inzwischen eine Skriptsprache erhalten haben.

Die Bedeutung der Skriptsprachen erkennt man auch daran, dass sich inzwischen einige Skriptsprachen etabliert haben, die noch leistungsfähiger sind. Am bekanntesten dürfte die Sprache Perl sein, die für jede UNIX Umgebung leicht zu bekommen sein müsste. Wer viel mit Skripten arbeitet, sollte sich diese Sprache unbedingt ansehen.

An dieser Stelle wird die Programmierung der Bourne Shell respektive der POSIX Shell diskutiert. Sie hat den Vorteil, dass sie auf jedem System vorhanden ist. Die Korn Shell und die `bash` sind weitgehend kompatibel. Lediglich die C-Shell unterscheidet sich erheblich in der Syntax.

-
- [Erstellen und Start eines Shellskripts](#)
 - [Variablen](#)
 - [Ablaufsteuerung](#)
 - [Die Unterscheidung: if](#)
 - [Bedingungen](#)
 - [Rückgabewert von Programmen](#)
 - [Die Fallunterscheidung: case](#)
 - [Die while-Schleife](#)
 - [Die for-Schleife](#)
 - [Funktionen](#)
 - [Ein- und Ausgaben aus dem Skript](#)
 - [Start und Umgebung von Skripten](#)

Erstellen und Start eines Shellskripts

Um ein Shellskript zu erzeugen, starten Sie einen Editor und führen einfach ein paar Kommandos hintereinander zeilenweise auf. Beispielsweise steht in der Datei namens skripttest:

```
# Mein erstes Skript
echo "ach ja"
ls
echo "soso"
```

Das Kommentarzeichen in einer Skriptdatei ist das #. Alles, was in der gleichen Zeile dahinter steht, geht den Interpreter nichts an. Die nächste Zeile ist der echo-Befehl, der einfach nur wiedergibt, was ihm als Parameter mitgegeben wird. Hier sagt er »ach ja«. Danach wird durch den Befehl `ls` der Inhalts des aktuellen Verzeichnisses angezeigt und schließlich kommentiert das Ganze der Befehl `echo` mit »soso«.

Um die Textdatei wie ein Programm aufrufen zu können, reicht es, sie mit dem Befehl `chmod 755` ausführbar zu machen (siehe S. `chmod`). Anschließend können Sie ihn durch Eingeben des Dateinamens direkt starten.

```
gaston> chmod 755 skripttest
gaston> skripttest
ach ja
skripttest
soso
gaston>
```

Variablen

Die Shellskripten können mit Variablen arbeiten. Variablen sind Speicher für Daten, die über einen Namen angesprochen werden. Die Variablen der Shell entsprechen den bereits bekannten Umgebungsvariablen. Der Unterschied liegt darin, dass Umgebungsvariablen an Kindprozesse weitergegeben werden. Mit dem Kommando `export` werden Shellvariablen zu Umgebungsvariablen. Der Name von Variablen beginnt mit einem Buchstaben und kann dann beliebig viele Ziffern und Buchstaben enthalten. Auch der Unterstrich ist zulässig.

Variablen werden gefüllt, indem Sie dem Variablennamen ein Gleichheitszeichen und den Wert folgen lassen. Dabei darf kein Leerzeichen zwischen der Variablen, dem Gleichheitszeichen und dem zugewiesenen Wert stehen. Um sich auf den Inhalt einer Variablen zu beziehen, stellen Sie dem Variablennamen ein Dollarzeichen voran. Zur Ausgabe von Variablen wird der Befehl `echo` verwendet, der seine Argumente auf die Standardausgabe ausgibt.

```
INFO="Tolles wetter!"  
echo
```

Bestimmte Variablennamen sind vorbelegt. Beispielsweise erfährt die Shell über besondere Variablen, wie der Aufruf des Skripts aussah. Mit `$0` erhalten Sie den Dateinamen, unter dem das Skript gestartet wurde. `$1` ist der erste Parameter, `$2` der zweite und so weiter. Aus der Variablen `$#` erfährt das Skript, mit wievielen Parametern es aufgerufen wurde. Beispiel:

```
[Aufrufparameter lesen]  
# Skript dreh - Tauscht Parameter  
echo "Ich heiße " $0 " und habe " $# "Parameter"  
echo $2 $1
```

Das Skript soll `dreh` heißen. Nachdem Sie mit `chmod` die Rechte auf ausführbar gesetzt haben, können Sie das Programm aufrufen. Das Ergebnis sieht so aus:

```
arnold@gaston > dreh anton erna  
Ich heiße ./dreh und habe 2 Parameter  
erna anton  
arnold@gaston >
```

Zusammenfassung der vordefinierten Variablen:

[Spezielle Variablen]
\$1 \$2 .. & Parameterstrings
\$0 & Name der Skriptdatei
\$# & Anzahl der übergebenen Parameter

Zuweisungen

Eine Variable wird definiert, wenn sie das erste Mal benutzt wird. Typischerweise geschieht dies durch eine Zuweisung. Shellvariablen enthalten Zeichenketten, die als Konstanten durch Anführungszeichen

begrenzt werden. Damit auch Sonderzeichen aufgenommen werden können, wird eine Zeichenkette in Hochkommata eingeschlossen. Das Zuweisungszeichen ist das Gleichheitszeichen. Beispiele:

```
a="a ist ein schlechter Name für eine Variable: zu kurz"
dieZahlPi=3.14
```

Wenn Sie die Befehle `echo $a` oder `echo $dieZahlPi` aufrufen, so ist die Ausgabe genau das, was Sie den Variablen zugewiesen haben. Bevor aber zu hohe Erwartungen entstehen, ist zu sagen, dass `dieZahlPi` keineswegs eine Zahl ist. Sie ist nur die Zeichenkette »3.14«.

Mit ganzen Zahlen können Sie in Skripten rechnen. Dazu gibt es den Befehl `expr`. Der Befehl erwartet als Parameter einen numerischen Ausdruck wie etwa `3 + 3`. Dabei ist darauf zu achten, dass zwischen den Zahlen und den Operanden ein Leerzeichen steht. Das Ergebnis solcher Rechenkünste können Sie einklammern, ein `$` davorstellen und es einer Variablen zuweisen. Das sieht dann so aus:

```
a=$(expr 3 + 3)
b=a * 3)
```

Wenn Sie sich per `echo` die Variablen `a` und `b` anschauen, stellen Sie fest, dass `a` nun 6 und `b` 18 ist. Bei genauem Hinsehen sehen Sie, dass bei der Zuweisung kein Leerzeichen und hinter `expr` überall Leerzeichen stehen. An dieser Stelle ist `expr` äußerst pingelig.

Natürlich fällt der Backslash vor dem `*` auf. Der ist notwendig, weil der Stern ja ein Sonderzeichen für die Shell ist. Damit der Stern den Befehl `expr` auch als Stern erreicht, müssen Sie eben einen Backslash davor setzen. In Tabelle steht, welche Operanden verwendet werden können.

[Operatoren in `expr`]
C|L Zeichen & Bedeutung

+ & Addition

- & Subtraktion

* & Multiplikation

/ & Division

% & Modulo: Rest einer Division

Die verwendbaren Operatoren in Shellskripten unterscheiden sich wenig von denen anderer Programmiersprachen.

Wie bereits im Kapitel über die Shell (siehe S. 1et) gezeigt wurde, gibt es bei den moderneren Nachfolgern der BourneShell den Befehl `let`, mit dem das Rechnen etwas eleganter wird als der Aufruf von `expr`. Da die Schreibweise etwas leichter lesbar ist, wird man diese Variante bevorzugen, sofern sichergestellt ist, dass die Shell der Zielsysteme den Befehl `let` verstehen. Hier noch einmal ein paar Beispiele:

```
gaston> let wert=45+5
gaston> echo
$wert
50
gaston> let wert=16\%5
gaston> echo$
wert
1
gaston> let wert=(1+3)*2
```

```
gaston> echo  
$wert  
8  
gaston>$
```

Unterabschnitte

- [Zuweisungen](#)
-

« [Erstellen und Start eines](#) | **[Programmierung von Shellskripten](#)** | [Ablaufsteuerung](#) »

Ablaufsteuerung

Werden die Kommandos einfach nur hintereinander ausgeführt, spricht man von einem Batchlauf. Richtig interessant werden die Skripten aber, wenn sie auf äußere Umstände reagieren können oder Arbeitsabläufe wiederholt werden können. Dazu dienen die Kommandos der Ablaufsteuerung. Dazu gehören Unterscheidungen und Schleifen.

Unterabschnitte

- [Die Unterscheidung: if](#)
 - [Bedingungen](#)
 - [Rückgabewert von Programmen](#)
 - [Die Fallunterscheidung: case](#)
 - [Die while-Schleife](#)
 - [Die for-Schleife](#)
 - [Funktionen](#)
-

Die Unterscheidung: if

Das Kommando `if` prüft eine Bedingung und führt den hinter dem `then` angegebenen Kommandoblock nur aus, wenn die Bedingung zutrifft. Der Interpreter kennt neben dem `if` auch den Befehl `else`. Darunter wird ein Kommandoblock zusammengefasst, der im anderen Fall ausgeführt wird, wenn also die Bedingung nicht zutrifft. Einen `else`-Zweig müssen Sie allerdings nicht bilden. Das Kommando wird durch `fi`, also ein umgedrehtes `if` abgeschlossen. Die Struktur einer Unterscheidung sieht so aus:

```
if Bedingung
then
  Befehle
else
  Befehle
fi
```

Die Befehle können einfach mehrere Programmaufrufe hintereinander sein. Es können aber durchaus auch wieder Kommandos zur Steuerung des Programmablaufs sein. Sie können Ablaufsteuerungskommandos also verschachteln.

Informationen zu `if` finden sich normalerweise auf der Manpage der jeweiligen Shell. Bei der `bash` können Sie aber auch `help` verwenden. Ohne Parameter werden die verfügbaren Befehle aufgezählt. Geben Sie als Parameter beispielsweise `if` an, wird der Syntax und die Verwendung von `if` erläutert.

Bedingungen

Für die meisten Strukturbefehle muss eine Bedingung abgefragt werden. Dazu gibt es den Befehl `test`. Er liefert einen Wahrheitswert in Abhängigkeit seiner Parameter zurück. Dabei gilt

[Das Kommando `test`]L|L Ausdruck & Wirkung

`test -f Name` & ist `Name` eine existierende Datei?

`test -d Name` & ist `Name` ein existierendes Verzeichnis?

`test String` & ist `String` eine nichtleere Zeichenkette?

`test String1 = String2` & ist Zeichenkette `String1` und `String2` gleich?

`test String1 != String2` & ist Zeichenkette `String1` und `String2` ungleich?

`test Nummer1 -eq Nummer2` & ist Zahl `Nummer1` gleich `Nummer2`?

`test Nummer1 -ne Nummer2` & ist Zahl `Nummer1` ungleich `Nummer2`?

`test Nummer1 -ge Nummer2` & ist Zahl `Nummer1` gr"o"ser oder gleich `Nummer2`?

`test Nummer1 -gt Nummer2` & ist Zahl `Nummer1` gr"o"ser `Nummer2`?

`test Nummer1 -le Nummer2` & ist Zahl `Nummer1` kleiner oder gleich `Nummer2`?

`test Nummer1 -lt Nummer2` & ist Zahl `Nummer1` kleiner `Nummer2`?

Das Beispielskript `dreh` wird nun erweitert. Das Skript soll feststellen, ob es mit der richtigen Anzahl Parameter aufgerufen wurde.

```
[Parameterzahl prüfen]
# Skript dreh - Tauscht Parameter
echo "Ich heiÙe " $0 " und habe "
    $# "Parameter"
if test$
    # -eq 2
then
    echo 1
else
    echo "Falsche Parameterzahl"
fi
```

Da die Schreibweise mit dem Kommando `test` sehr gewöhnungsbedürftig für Benutzer anderer Programmiersprachen ist, gibt es dafür eine alternative Schreibweise. Dabei wird das Wort `test` durch eine eckige, öffnende Klammer ersetzt, die nach dem letzten Parameter von `test` wieder geschlossen wird. Das liest sich komplizierter als es ist. Das Beispiel macht es deutlich:

```
[Rechteckige Klammer statt test]
# Skript dreh - Tauscht Parameter
echo "Ich heiÙe "
    $0 " und habe "$
    # "Parameter"
if [
    $# -eq 2 ]
then
    echo$
    2
    $1
else
    echo "Falsche Parameterzahl"
```

fi\$

Eine kleine Stolperfalle möchte ich Ihnen nicht vorenthalten. Im folgenden Skript soll eine Diskette formatiert werden, wenn der Parameter »new« angegeben wird. Das Abfragen des Parameters ist fast das gleiche wie beim Abfragen der Anzahl der Parameter.

```
[Harmlose Abfrage?]
if [ $1 = "new" ]
then
    echo "Formatiere..."
fi
echo "Und los gehts..."
```

Nun rufen Sie das Skript dreimal auf! Einmal mit dem Parameter new, dann mit dem Parameter old und schließlich ohne Parameter. Und da gibt es eine Überraschung!

```
gaston> trick new
Formatiere...
Und los gehts...
gaston> trick old
Und los gehts...
gaston> trick
./trick: [: =: unary operator expected
Und los gehts...
gaston>
```

Die Fehlermeldung sagt aus, dass der Operator = zwei Operanden erwartet hätte, aber nur einer da war. Tatsächlich wird \$1 vor dem Vergleich ausgewertet und da dort nichts drin steht, ist zwischen der eckigen Klammer und dem Gleichheitszeichen nichts. Es ist also so, als würde dort folgender Ausdruck stehen:

```
if [  = "new" ]
```

Sie können solche Überraschungen vermeiden, indem Sie die Variablen in Anführungszeichen stellen. Wie bereits an anderer Stelle erwähnt, bewirken die Anführungszeichen das Zusammenfassen mehrerer Worte zu einem Parameter, aber es lässt im Gegensatz zu den Hochkommata die Auswertung der Variablen zu.

```
[Sichere Abfrage]
if [ "$1" = "new" ]
then
    echo "Formatiere..."
fi
echo "Und los gehts..."$
```

Auch wenn die Variablenauswertung von \$1 leer ist, steht nun links vom Gleichheitszeichen etwas, nämlich zwei Anführungszeichen, also:

```
if [ "" = "new" ]
```

Rückgabewert von Programmen

Aufgerufene Programme liefern einen Wert zurück, der über den Erfolg oder Misserfolg ihrer Tätigkeit Auskunft geben. Dabei ist es unter UNIX Standard, dass ein fehlerfrei gelaufenes Programm eine 0 zurückgibt. Diese wird dann von `if` als wahr interpretiert. Liefert das Programm dagegen eine Zahl ungleich 0 zurück, deutet das auf einen Fehler hin. Darum interpretiert `if` ein Ergebnis ungleich 0 als falsch. Das ist für den C-Programmierer etwas ungewohnt, da dort eine 0 als falsch und alles andere als wahr interpretiert wird.

Einige dieser Programme sind geradezu prädestiniert für die Arbeit in Skripten. Der Befehl `cmp` vergleicht zwei Dateien. Wird er mit der Option `-s` aufgerufen, macht er dabei keine Ausgaben. Er gibt bei Gleichheit 0, ansonsten 1 zurück.

Die Fallunterscheidung: case

Die Fallunterscheidung ist eine Art gestaffeltes if. Der Inhalt einer Variablen wird untersucht und für jeden denkbaren Inhalt eine Aktion definiert. Eine Anwendung findet sich in den rc-Skripten, die beim Booten des Systems gestartet werden. Hier wird unterschieden, ob als Parameter die Worte »start« oder »stop« oder ein anderes Schlüsselwort übergeben wurde. Zwischen den Schlüsselworten case und in steht die Variable, deren Inhalt zur Unterscheidung führen soll. Ein Unterscheidungszweig beginnt mit einem Begriff, der bearbeitet werden soll. Diese Maske schließt mit einer rechten, runden Klammer und kann Platzhalter verwenden, wie sie bei der Namensauswahl durch die Shell üblich sind. Es folgen die Kommandos bis zu einem doppelten Semikolon, das das Ende eines Zweiges definiert. Die Fallunterscheidung endet mit dem Schlüsselwort esac.

```
case Variable in  
Maske) Kommandos ;;  
Maske) Kommandos ;;  
*) Default-Kommandos ;;  
esac
```

Das folgende Beispiel unterscheidet zwischen einfachen Begriffen. Es ist als der Grundbaustein eines deutsch-englischen Übersetzungsprogramms namens englisch gedacht. Wer mag, kann es ja erweitern.

```
[Intelligenter Übersetzer]  
case "$1" in  
    haus) echo "house" ;;  
    auto)  echo "car"  ;;  
    kind)  echo "child" ;;  
    *)    echo "stuff" ;;  
esac
```

Der Stern als letzte Maske dient dem Abfangen aller der Begriffe, die durch die davor stehenden nicht erfasst wurden. Als nächstes Beispiel wird ein Skript erstellt, der meinname heißen soll und Namen analysieren soll. Der Name wird dem Skript als Parameter mitgegeben. Damit wird die Verwendung von Masken gezeigt.

```
[Grüß-August]  
case "  
$1" in  
    [ww]illemer) echo "verwandschaft!!!!" ;;  
    A* | a*)    echo "welch ein Name!" ;;  
    *)          echo "soso! Nett, Sie kennenzulernen" ;;  
esac$
```

Die erste Zeile besagt, dass der erste Parameter untersucht wird. In der zweiten Zeile sehen Sie, dass das Skript in der Lage ist, den Namen Willemer zu erkennen. Dem Skript ist dabei gleich, ob das W klein oder groß geschrieben wird. In der nächsten Zeile werden Namen bearbeitet, die mit einem A anfangen. Der senkrechte Strich bedeutet ODER. Also ist es wieder egal, ob es ein kleines oder ein großes A ist. Der Stern allein ist der Default, wenn keines der bisherigen Muster gegriffen hat.

Die while-Schleife

Schleifen ermöglichen es, Abläufe zu beschreiben, die sich wiederholen. Damit ein definiertes Ende stattfindet, läuft die Schleife nur so lange, wie eine Bedingung eingehalten wird. Diese Bedingung sollte sorgfältig gewählt werden, sonst kommt es zur gefürchteten Endlosschleife. Das bedeutet, dass das Programm bis zum nächsten Stromausfall läuft.

while Bedingung
do
Befehle
done

Als Beispiel für eine solche Schleife sollen alle Parameter darauf überprüft werden, ob sie mit einem Minuszeichen beginnen. Dann sollen sie als Option gelten. Ansonsten handelt es sich um ein Argument. Für diese Aufgabe werden nun zwei Ablaufsteuerungen ineinander verschachtelt. Da eigentlich nur ein Abfragefall existiert, würde man zunächst an die `if` Bedingung denken. In diesem Fall macht man sich aber die Fähigkeit des `case` zunutze, Muster zu erkennen.

```
[Optionserkennung]
while test -n "$1"
do
    case
    $1 in
        -*) echo "Option:$
1" ;;
        *) echo "Argument:
$1" ;;
    esac
    shift # schiebt die Parameter einen weiter
done$
```

Es wird immer der erste Parameter abgefragt. In der Schleife befindet sich aber der Befehl `shift`. Dieser schiebt die Übergabeparameter durch. Der erste Parameter verschwindet und alle anderen rücken einen nach. Irgendwann wird so der erste Parameter leer. Zur Veranschaulichung zeigt die folgende Tabelle die Variablen `$1` bis `$5`. Jede neue Zeile zeigt die Parameter nach einem weiteren `shift`.

```
[Parameter und shift]L|L|L|L|L $1 & $2 & $3 & $4 & $5
anton & berta & caesar & dora & emil
berta & caesar & dora & emil &
caesar & dora & emil & &
dora & emil & & &
emil & & & &
& & & &
```

In der Bedingung der Schleife ist die Variable in Anführungszeichen gesetzt. Dadurch wird verhindert, dass ein Aufruf ohne Parameter zu einem Fehler führt. Beim `case` können sie weggelassen werden, da diese Position nie erreicht wird, wenn im ersten Parameter nichts steht.

Schleifen können durch das Kommando `break` unterbrochen werden. Dieser Befehl steht typischerweise hinter einer `if`-Konstruktion. Allerdings kann dieser Befehl leicht zu etwas undurchsichtigem Code führen. Besser ist es, die vollständige Bedingung für das Durchlaufen einer Schleife direkt hinter dem `while` zu formulieren. In die gleiche Kategorie gehört der Befehl `continue`, der dazu führt, dass der Rest des Schleifenkörpers nicht ausgeführt wird, sondern sofort zur Abfrage am Kopf gesprungen wird.

« [Die Fallunterscheidung: case](#) | **[Ablaufsteuerung](#)** | [Die for-Schleife](#) »

Die for-Schleife

Die for-Schleife ist spezialisiert auf die Abarbeitung von Listen.

```
for Variable in Liste
do
Kommandos
done
```

In der Schleife wird die Variable die Werte, die durch die Liste hinter dem Schlüsselwort `in` definiert ist, der Reihe nach annehmen. Auf die Variable kann innerhalb der Schleife durch ein vorangestelltes Dollarzeichen (\$) zugegriffen werden. Die Schleife wird so oft durchlaufen, wie Argumente hinter dem 'in' stehen. Dabei nimmt die Variable nacheinander jedes der Argumente als Inhalt an. Beispiel:

```
[Lieblingsfarben]
for i in blau gelb grün rot
do
  echo "Meine Lieblingsfarbe ist $i. Also fahre ich"
  "$i"e Autos."
done$
```

Die Ausgabe der Schleife ist:

```
Meine Lieblingsfarbe ist blau. Also fahre ich blaue Autos.
Meine Lieblingsfarbe ist gelb. Also fahre ich gelbe Autos.
Meine Lieblingsfarbe ist grün. Also fahre ich grüne Autos.
Meine Lieblingsfarbe ist rot. Also fahre ich rote Autos.
```

Hier wird noch einmal demonstriert, dass `$i` auch innerhalb der Anführungszeichen interpretiert wird. Wollen Sie erreichen, dass der Inhalt der Zeichenkette nicht interpretiert wird, müssen Sie Hochkommata verwenden.

Besonders interessant wird die for-Schleife, wenn statt einer festen Liste von Zeichenketten Dateien verwendet werden, die über Wildcards ausgewählt werden. Das folgende Beispiel wandelt Audiodateien in mp3"-Dateien um und löscht anschließend die Originaldateien.

```
[wav nach mp3 konvertieren]
for i in *.wav
do
  notlame $i `basename
  $i .wav`.mp3
  rm$
i
done
```

Eine kleine Schönheitsoperation wurde hier noch mit dem Kommando `basename` durchgeführt. `basename` entfernt den Verzeichnisnamen einer Datei. Wird noch ein weiterer Parameter außer dem Dateinamen angegeben, wird dieser als Anhängsel betrachtet, der von hinten abgeschnitten werden soll. Im Beispiel wird von `$i` der Anhang `.wav` abgeschnitten. An das Ergebnis dieser Operation wird

mp3 angehängt und das Ganze als Zielfilei des Programmes notlame verwendet. Angenommen im aktuellen Verzeichnis gäbe es die Dateien a.wav, b.wav und c.wav, so wird die Schleife folgende Befehle erzeugen:

```
notlame a.wav a.mp3  
rm a.wav  
notlame b.wav b.mp3  
rm b.wav  
notlame c.wav c.mp3  
rm c.wav
```

« [Die while-Schleife](#) | [Ablaufsteuerung](#) | [Funktionen](#) »

Funktionen

In Shellskripten können Sie Funktionen definieren, die einen bestimmten Arbeitsablauf zusammenfassen und von anderer Stelle im Skript beliebig oft aufrufen. Auf diese Weise wird einerseits das Skript übersichtlicher und andererseits auch kürzer, da Sie Codesequenzen, die sich wiederholen, zusammenfassen können.

```
[Simple Funktion]
meinefunktion()
{
    echo "ich tue hier etwas"
}
```

```
meinefunktion
```

Hier wird die Funktion namens `meinefunktion()` definiert und später einfach direkt über ihren Namen aufgerufen.

Es ist auch möglich, Parameter an Funktionen zu geben. Der Mechanismus entspricht dem bei der Parameterübergabe an Skripten und deren Auswertung. Beim Aufruf werden die Parameter einfach durch Leerzeichen getrennt hinter dem Funktionsaufruf aufgelistet. Innerhalb der Funktion wird auf die Parameter mit den Variablen `$1`, `$2` und so weiter zugegriffen. Hier ein Beispiel:

```
[Funktion mit Parameterübergabe]
meinefunktion()
{
    echo $1
    echo $2
}
```

```
meinefunktion "huhu"
meinefunktion "haha" 12
```

Der Aufruf des Shellskripts bringt folgende Ausgaben auf dem Bildschirm.

```
gaston> func
huhu
```

```
haha
12
gaston>
```

Ein- und Ausgaben aus dem Skript

Die einfachste Art der Ausgabe erfolgt durch Aufruf des Befehls `echo`. Er gibt seine Parameter über die Standardausgabe aus.

Wenn Sie allerdings Texte größeren Umfangs ausgeben wollen, wie beispielsweise Hilfetexte, dann wird die Verwendung von `echo` etwas mühsam. Sie können dann dem Befehl `cat` die Eingabedatei aus dem Shellskript heraus geben. Und `cat` wird das tun, was es immer tut, nämlich die Datei auf der Standardausgabe ausgeben. Im Beispiel sieht das so aus:

```
[schwatzhaft]
cat «!
Hier steht nun eine sehr weitschweifige Erklärung, wie das
Programm zu benutzen ist, wer der geniale Programmierer
dieser Zeilen ist und dass man nach Benutzung dieses
Skriptes nie wieder ein anderes ansehen wird.
!
```

Der Text muss durch ein Zeichen eingeklammert werden, das im Text selbst natürlich nicht vorkommen darf. Das Ganze wird dann mit zwei Kleinerzeichen in den Befehl `cat` geschoben. Auf diese Weise kann jedem Programm die Standardeingabe zugeschoben werden, nicht nur `cat`.

Neben der Ausgabe kann es erforderlich sein, vom Anwender Eingaben zu erfragen. Dazu gibt es das Kommando `read`, das als Parameter die Variable hat, in die die Eingabe gelangt.

```
read ANSWER
```

Nach der Eingabe mit einem abschließenden Return wird die Eingabezeile in der Variablen `ANSWER` stehen. Im Gegensatz zu Perl gelangt das Return nicht in die Variable.

Start und Umgebung von Skripten

Angenommen in der Datei skripttest steht ein Shellskript, das nur darauf wartet, ausgeführt zu werden. Um es zu starten, können Sie verschiedene Wege gehen. Der erste Weg wurde schon beschrieben. Sie ändern die Dateirechte auf ausführbar und starten es wie ein normales Programm.

```
gaston> skripttest
```

Der zweite Weg, ein Skript auszuführen, ist indem Sie eine Shell aufrufen und ihr die Datei als Parameter übergeben. Tatsächlich sind diese beiden Arten, ein Shellskript zu starten, äquivalent. Auch bei dem direkten Start des Dateinamens wird eine neue Shell gestartet, die die Datei skripttest interpretiert.

```
gaston> sh skripttest
```

Mit dem Kommando `.` (Punkt) und dem Dateinamen kann die Datei skripttest von der aktuellen Shell ausführen lassen.

```
gaston> . skripttest
```

Der Unterschied zwischen dem Aufruf per Punkt oder per Subshell ist wichtig, da ein mit Punkt aufgerufenes Skript keine neue Shell startet. Stattdessen interpretiert die laufende Shell das Skript. Nur so kann ein Skript den Zustand der aktuellen Shell verändern kann. Wechselt das Skript das Verzeichnis, legt es Umgebungsvariablen an oder verändert es sie, ist dies nach Ausführung des Skripts in der Arbeitshell erfolgt. Wird dagegen eine Tochtershell gestartet, wirken sich die Änderungen nur dort aus und hat auf die aktuelle Shell keinerlei Auswirkungen. Anders ausgedrückt: Soll ein Skript Variablen setzen, die in der aktuellen Shell später gebraucht werden, muss dieses Skript zwingend mit dem Punkt aufgerufen werden.

Die Variablen einer Shell werden normalerweise nicht an Kindprozesse weitergegeben. Wenn in einem Skript eine Variable gesetzt wird, die von einem Kindprozess gelesen werden soll, muss die Variable exportiert werden. Beispiel:

```
MYENV="Tolles Wetter"  
export MYENV
```

In der Kornshell und der bash kann das Setzen und Exportieren der Variable in einem Kommandoschritt ausgeführt werden.

```
export MYENV="Tolles Wetter"
```

Da dies nicht unter der Standard Bourne-Shell funktioniert und Skripten sehr oft aus Kompatibilitätsgründen mit der `/bin/sh` gestartet werden, ist es sicherer, Zuweisung und Export getrennt zu halten.

In der ersten Zeile eines Skripts kann festgelegt werden, welche Shell bzw. welcher Interpreter für diese Datei geladen werden soll. Das erste Zeichen hinter dem Kommentarzeichen ist ein Ausrufezeichen. Dann folgt mit komplettem Pfad der Interpreter. Beispiel:

```
#!/bin/sh
```

Dies ist wichtig, weil das Skript eventuell von jemandem gestartet wird, der vielleicht die C-Shell verwendet, die eine andere Syntax hat.

« [Ein- und Ausgaben aus](#) | [Programmierung von Shellskripten](#) | [Perl](#) »

Perl

Larry Wall hat 1987 die Sprache Perl entwickelt, um die UNIX Administration zu vereinfachen. Perl hat inzwischen Karriere im Internet gemacht, weil es auf dem Gebiet der Auswertung von Texten kaum noch zu schlagen ist.

Perl ist eine Interpretersprache, die umfangreiche Möglichkeiten im Umgang mit Texten und Mustern liefert und dabei dennoch systemnahe Programmierung ermöglicht. Die Sprache wurde von Larry Wall als Hilfsmittel zur Administration geschaffen und als freie Software unter die GNU General Public Licence gestellt. In der Administration ist Perl inzwischen ein gängiges Werkzeug. Im Bereich der CGI-Skripten ist Perl durch seine Fähigkeiten mit Texten zu hantieren fast konkurrenzlos. Da es den Interpreter inzwischen auf beinahe jeder Plattform gibt, ist es ein ideales Tool, um portable Abläufe zu programmieren. Mit Perl können die UNIX Systemaufrufe erreicht werden, man kann mit Tk grafisch bedienbare Programme schreiben und über entsprechende Schnittstellen auf Datenbanken zugreifen.

Der Syntax ist an C, die Shellskripten und die bekannten UNIX-Tools angelehnt.

-
- [Interpreter und Skript](#)
 - [Variablen](#)
 - [Skalare](#)
 - [Variablennamen](#)
 - [Operationen auf Skalare](#)
 - [Arrays](#)
 - [Hash](#)
 - [Interaktiv](#)
 - [Ein- und Ausgabe](#)
 - [Aufrufparameter](#)
 - [Umgebungsvariablen](#)
 - [Ablaufsteuerung](#)
 - [Bedingungen](#)
 - [if](#)
 - [for](#)
 - [foreach](#)
 - [Sonstige Schleifen: while und until](#)
 - [Funktionen](#)
 - [Dateien](#)
 - [Schreiben und Lesen](#)

- Umgang mit Dateien
 - Perl und UNIX
 - Aufruf von UNIX-Programmen
 - UNIX Systemprogrammierung
 - Grafische Oberfläche: Tk
 - Widgets und Ressourcen
 - Kontrollelemente
 - Widgetanordnung
 - Informationsquellen
-

« Start und Umgebung von | **Programmierung** | Interpreter und Skript »

Interpreter und Skript

Der Interpreter von Perl heißt einfach `perl` und befindet sich im Verzeichnis `/usr/bin`. Das ist deswegen so wichtig, weil in einem Perlskript angegeben wird, wie der Interpreter lautet. Dazu schreiben Sie in die erste Zeile eines Perlskripts:

```
#!/usr/bin/perl -w
```

Etwas ungewöhnlich ist die Option `-w`. Sie zeigt alle Warnungen in ausführlicher Form an. Die Dokumentation weist intensiv darauf hin, diese Option zu verwenden.

Anschließend wird die Datei mit Hilfe des Befehls `chmod` (siehe S. `chmod`) ausführbar gemacht. Danach kann das Skript direkt von der Kommandozeile aus aufgerufen werden.

Alternativ kann ein Skript natürlich auch direkt als Parameter für den Interpreter aufgerufen werden.

```
perl -w skript.pl
```

Als Endung für die Skripte verwendet man traditionell `.pl`, das ist allerdings keineswegs zwingend. Insbesondere bei CGI-Skripten geht es den Anwender nichts an, in welcher Sprache sie realisiert sind. Und dort ist es eher üblich, dass man die Endung weglässt.

Wie unter UNIX üblich, unterscheidet Perl genau zwischen Groß- und Kleinschreibung.

Variablen

Perl besitzt wie alle Programmiersprachen Variablen. Allerdings sind deren Typen etwas ungewöhnlich. Aber gerade das ist der Grund, warum Perl so extrem leistungsfähig ist. Um die verschiedenen Variablentypen zu unterscheiden, wird jeder Variablen ein Sonderzeichen vorangestellt.

Unterabschnitte

- Skalare
 - Zahlenkonstanten
 - Zeichenketten
 - Variablennamen
 - Strenge Aufsicht
 - Operationen auf Skalare
 - Numerische Operationen
 - Operatoren auf Zeichenketten
 - Arrays
 - Hash
-

Unterabschnitte

- [Zahlenkonstanten](#)
- [Zeichenketten](#)

Skalare

Skalare nennt Perl die einfachen Variablen, die sowohl numerische Werte als auch Strings aufnehmen können. Enthält eine Variable eine Zahl, wo eine Zeichenkette erwartet wurde, wandelt Perl diese kurzentschlossen um. Auch umgekehrt wird eine Zeichenkette automatisch in eine Zahl umgewandelt, wenn eine Zahl erwartet wird. Ist der Inhalt nicht als Zahl zu interpretieren, ist ihr Wert eben 0.

Zahlenkonstanten

Zahlenkonstanten können ganze Zahlen oder Fließkommazahlen sein. Zur Darstellung des Dezimalkommas verwendet Perl, wie im englischen Sprachraum üblich, den Punkt. Große Zahlen oder kleine Brüche stellt Perl mit Hilfe des vom Taschenrechner bekannten Buchstaben e dar. Hinter dem e steht der Exponent zur Basis 10, mit dem die Zahl zu multiplizieren ist. Um es einfacher zu sagen, steht dort die Anzahl der Stellen, um die das Komma nach rechts zu verschieben ist. Die folgenden Zahlen sind alle gleich groß.

```
1.000000e6
10.00000e5
100.0000e4
1000.000e3
10000.00e2
100000.0e1
10000000
100000000e-1
1000000000e-2
```

Auch hexadezimale und oktale Zahlen können verwendet werden. Mit oktalen Zahlen sind Sie schon einmal beim Befehl `chmod` (siehe S. `chmod`) in Berührung gekommen. Eine oktale Zahlendarstellung wird durch eine führende 0 eingeleitet. 010 ist also nicht 10 sondern 8. Die hexadezimale Darstellung beginnt mit 0x. 0x10 ist also 16. Wenn Sie nicht wissen, was hexadezimal oder oktal ist, brauchen Sie sich keine Sorgen zu machen. Dann brauchen Sie auch deren Darstellung nicht. Sie sollten dann lediglich aufpassen, dass Sie keine Zahl mit 0 beginnen lassen.

Sehr praktisch ist die Möglichkeit, in große Zahlen Unterstriche einzufügen, ohne den Zahlenwert zu verändern. So kann man Unterstriche der besseren Lesbarkeit an Stelle der im Deutschen üblichen Tausenderpunkte zu verwenden. Eine Million ist dann im Quelltext als `1_000_000` leichter zu erkennen als in der normalen Darstellung `1000000`.

Zeichenketten

Zeichenkettenkonstanten werden in Anführungszeichen oder Hochkommata eingeschlossen. Der Unterschied zwischen beiden ist, dass innerhalb von Anführungszeichen Variablen ausgewertet

werden und bei Hochkommata nicht.

In einer Zeichenkette gibt es für diverse Sonderzeichen spezielle Zeichen, die alle mit einem Backslash beginnen. Diese haben folgende Bedeutung:

[Sonderzeichen in Zeichenketten]L|L Symbol & Wirkung

n & Line Feed

t & Tabzeichen

\$ & Dollarzeichen

& Backslash-Zeichen

l & nächstes Zeichen in Kleinbuchstaben wandeln

L .. E & Zeichen zwischen L und E in Kleinbuchstaben wandeln

u & nächstes Zeichen in Großbuchstaben wandeln

U .. E & Zeichen zwischen U und E in Großbuchstaben wandeln

« [Variablen](#) | [Variablen](#) | [Variablennamen](#) »

Unterabschnitte

- [Strenge Aufsicht](#)

Variablennamen

Die Variablennamen von Skalaren werden durch ein \$ eingeleitet. Danach muss ein Buchstabe oder ein Unterstrich erscheinen und anschließend können in beliebiger Folge Buchstaben, Ziffern oder Unterstriche folgen. Ein Variablenname kann maximal 255 Buchstaben lang sein. Mein Tipp: schöpfen Sie es nicht bis zum letzten aus. Allerdings sollten Sie sich auch nicht auf zwei oder drei Buchstaben beschränken. Verwenden Sie so viele, dass klar wird, was die Variable enthält. Das folgende Beispiel ist auch für den Uneingeweihten leicht lesbar.

```
[Sprechende Variablennamen]
$rechnungsbetrag = $stunden * $stundensatz;
$mwstbetrag = $rechnungsbetrag * $mwstsatz;
```

Dagegen werden Sie die folgenden zwei Zeilen vermutlich bereits in einem halben Jahr selbst nicht mehr verstehen ohne sich enorm zu konzentrieren. Da Sie sich schon genug auf das Programmieren konzentrieren müssen, ersparen Sie es sich bei der Wahl der Variablen!

```
[Übelster Abkürzfümmel]
$rgb = $std * $ss;
$mb = $rgb * $ms;
```

Strenge Aufsicht

Obwohl es ein schönes Gefühl ist, nicht kontrolliert zu werden, führt die mangelnde Kontrolle durch den Interpreter auch leicht zu Flüchtigkeitsfehlern. Der erste Schritt zu sicheren Programmen ist die Verwendung der Option -w hinter dem Interpreteraufruf. Damit wird beispielsweise erreicht, dass eine Variable angemahnt wird, die im Quellcode nur einmal vorkommt. Man kann relativ sicher sein, dass eine solche Variable aufgrund eines Schreibfehlers entstanden ist.

Ein weiterer Schritt zur Steigerung der Sicherheit wird durch die Anweisung `use strict;` erreicht. Wenn diese zu Anfang des Skripts gesetzt wird, wird jede Variable darauf geprüft, ob sie vorher deklariert wurde. Eine Variable kann durch Voranstellen des Schlüsselwortes `my` deklariert werden. Die Einleitung zu dem Beispiel von oben würde dann lauten:

```
[Deklarationen]
#!/usr/bin/perl -w
use strict;
my $rechnungsbetrag;
my $stunden;
my $stundensatz;
my $mwstsatz;
my $mwstbetrag;
```

Der Mehraufwand, die Variablen ganz zu Anfang alle zu nennen, könnte den einen oder anderen Programmierer vielleicht sogar dazu verführen, hinter der Variablen einen Kommentar zu schreiben, wozu die Variable gebraucht wird. Das würde wiederum denjenigen glücklich machen, der das Programm später warten muss. Und derjenige, der das Programm später warten muss, könnte im schlimmsten Fall der Autor selbst sein.

« [Skalare](#) | [Variablen](#) | [Operationen auf Skalare](#) »

Unterabschnitte

- [Numerische Operationen](#)
- [Operatoren auf Zeichenketten](#)

Operationen auf Skalare

Oben wurde es schon ein wenig vorweggenommen. Die Variablen erhalten ihre Werte durch die Zuweisung. Das zuständige Zeichen ist das Gleichheitszeichen.

```
[Zuweisung]
$name = "hugo";
$wert = 15;
```

Numerische Operationen

Sie werden mit numerische Variablen auch rechnen wollen. Dazu werden die Operatoren verwendet, die sich seit den Zeiten von Fortran inzwischen in fast allen Programmiersprachen durchgesetzt haben. Dabei werden die folgenden Operatoren jeweils zwischen zwei Operanden gestellt, weshalb man sie auch binäre Operatoren nennt.

[Binäre numerische Operatoren]C|L Operator & Wirkung

- + & Addition
- & Subtraktion
- * & Multiplikation
- / & Division
- % & Modulo (Rest einer ganzzahligen Division)
- ** & Potenzierung

Die Priorität der Operatoren ist in der Tabelle nach unten ansteigend. Es gilt also die bekannte Punkt-vor-Strich-Regel. Natürlich gibt es auch in Perl Klammern, mit denen man die Regel außer Kraft setzen kann. Zu den wenigen unären Operatoren Unäre Operatoren sind solche, die nur auf einen Operanden wirken. gehört die Inkrementierung und Dekrementierung, die in C mit ++ bzw. - realisiert wird. Auch diese Operatoren gibt es in Perl. ++ bewirkt, dass der Inhalt der Variablen um 1 erhöht wird, und -, dass er um 1 vermindert wird.

Operatoren auf Zeichenketten

Durch einen Punkt werden zwei Zeichenketten aneinandergehängt.

```
$fullname = $firstname . " " . $lastname;
```

Mit einem x kann ein String vervielfältigt werden. Um das Wort »huhu« zu erzeugen, könnten Sie in Perl also schreiben:

```
$ruf = "hu" x 2;
```

Innerhalb einer Zeichenkette können Sie Zeichenketten ersetzen. Das Beispiel zeigt das Ersetzen jedes Auftreten von »saulus« nach »paulus« in der Variablen \$apg.

```
$apg = s/saulus/paulus/g;
```

Der Syntax scheint vertraut. Tatsächlich entspricht der Befehl dem, was man von `vi` und `sed` kennt. Das `g` am Ende bewirkt das Ersetzen aller Vorkommnisse. Wird es weggelassen, wird nur das erste Auftreten ersetzt. Die Suchzeichenkette ist ein regulärer Ausdruck, wie er ab `S. regexp` erläutert wird. Steht hinter dem `g` noch ein `i` wird Groß- und Kleinschreibung ignoriert.

Perl besitzt eine große Menge an Möglichkeiten, mit Texten umzugehen. Sie entfalten sich vor allem im Zusammenhang mit dem Array und dem Hash.

« [Variablennamen](#) | **Variablen** | [Arrays](#) »

Arrays

Ein Array ist eine Folge von Skalaren. Wird ein Array als Konstante dargestellt, sind es einfach Werte, die durch Kommata voneinander getrennt werden und von einer runden Klammer umgeben sind. Der Name einer Arrayvariablen, die eine solche Liste aufnimmt, wird mit dem Zeichen @ eingeleitet.

```
[Arrays]
my @primzahlen = (1,2,3,5,7,11,13,17,19);
my @namensliste = ("karl", "Friedrich", "Hans");
```

Die Werte innerhalb einer solchen Klammer müssen nicht gleichartig sein. Es können alle Skalare bunt gemischt auftreten. Sie müssen nicht einmal als Konstanten auftreten, es können auch Variablen eingesetzt werden.

Um auf ein Element einer Arrayvariablen zuzugreifen, wird der Variablen in eckigen Klammern der Index des referenzierten Elements angefügt. Der Index beginnt bei 0. Beim Referenzieren wird auf ein Element des Arrays und nicht auf das gesamte Array zugegriffen. Die Elemente eines Arrays sind aber Skalare. Konsequenterweise wird einem Element eines Arrays auch wieder ein \$ vorangestellt.

```
$wert = $primzahlen[4];
```

In diesem Fall wird das fünfte Element, also die 7 in die Variable \$wert kopiert. Es ist nicht das vierte Element, da die Indizes bei 0 beginnen. Perl kann aber auch mehrere Elemente eines Arrays in ein anderes Arrays kopieren. In diesem Fall, werden die einzelnen Indizes in der eckigen Klammer in der gewünschten Reihenfolge aufgezählt und durch Kommata getrennt.

```
@werte = @primzahlen[4,3,1];
```

Damit enthält das Array @werte den Wert (7, 5, 2). Wenn dieser Vorgang für den Programmierer anderer Sprachen schon ungewöhnlich wirkte, dann wird die Möglichkeit, mit einer Zuweisung Werte in mehrere Variablen zu schieben, zur äußerst praktischen Kuriosität.

```
($erst, $zweit, @neuprimzahlen) = @primzahlen;
```

Hier wird das erste Element des Arrays @primzahlen der Skalarvariablen \$erst, das zweite der Skalarvariablen \$zweit und der Rest des Arrays @primzahlen dem Array @neuprimzahlen zugewiesen. Das heißt, dass dem Array @neuprimzahlen die Werte 1 und 2 fehlen und ansonsten die restlichen Werte von @primzahlen hat.

Den umgekehrten Weg erreichen Sie mit der nützlichen Funktion split(). Mit ihr können Sie eine Zeichenkette in ein Array aufbrechen. Die Bruchstelle wird durch ein frei wählbares Zeichen festgelegt. Diese Funktion erleichtert beispielsweise die Analyse von CGI-Übergabedaten sehr. Das folgende Beispiel teilt einen Satz in seine Worte an den Stellen auf, wo ein Leerzeichen steht.

```
@wort = split(/ /, $satz);
```

Der erste Parameter ist das Leerzeichen, das durch beliebige Zeichen eingeschlossen werden kann. Üblicherweise verwendet man Anführungszeichen oder Schrägstriche. \$wort[0] hat anschließend das erste Wort von \$satz, \$wort[1] das zweite und so weiter.

Eine kleine Spielerei soll als Beispiel dienen. Mit der Funktion `split()` wird ein Namen an der Stelle, wo sich Leerzeichen befinden aufgespalten. Das erste Element ist dann der Vorname und das zweite ist der Nachname.

```
[Namenszerlegung]
#!/usr/bin/perl -w
$ask = <STDIN>;
chomp($ask); # wirf den zeilenvorschub weg!
@name = split(/ /, $ask);
print $name[0]."\n";
print $name[1]."\n";
```

Da es Menschen gibt, die mehr als einen Vornamen besitzen, ist die Ermittlung des Nachnamens nicht besonders zuverlässig. An sich brauchen wir nicht das zweite, sondern das letzte Element des Arrays. Hier die Änderung für die letzte Zeile.

```
print $name[$#name]."\n";
```

Die Zeichenkombination `$#` ermittelt den höchsten Index eines Arrays. Dieser Wert ist also um eins geringer als die Dimension des Arrays. Die beiden Zeichen werden dem Namen des Array statt des Zeichens `@` vorangestellt. Ein leeres Array liefert -1. Im Beispiel oben wird dies als Index für das letzte Element verwendet. Die Dimension eines Arrays kann auch ermittelt werden, indem eine direkte Zuweisung eines Arrays an einen Skalar durchgeführt wird. Da ein Skalar ein Array nicht aufnehmen kann, geht Perl davon aus, dass der Programmierer die Dimension ermitteln will.

« [Operationen auf Skalare](#) | [Variablen](#) | [Hash](#) »

Hash

Dieser Variablentyp ist eine Art paarweises Array, wobei das jeweils vordere Element als Zugriffsschlüssel auf das hintere wirkt. Man spricht auch von einem assoziativen Array. Beispiel:

```
[Hash]
%kfz = ('sl', 'Schleswig', 'fl', 'Flensburg', 'hh', 'Hamburg');
$kfz{'hg'} = 'Bad Homburg';
$kfz{'hg'} = 'Hochtaunuskreis';

$key="hg";
print "Schlüssel = \"$key\", wert = \"$kfz{\{$key\}}\n";
```

In der ersten Zeile wird das Hash namens `kfz` definiert. Das sieht ähnlich wie bei der Definition eines Arrays aus. Es fällt auf, dass die Variable ein Prozentzeichen trägt und die Art der Inhalte lässt vermuten, dass hier je zwei benachbarte Werte immer zusammen gehören. Der jeweils erste Wert ist der Schlüssel und der zweite der Inhalt. Der Zugriff auf ein Element des Hash ähnelt dem beim Array. Allerdings wird keine Nummer als Index verwendet, sondern der Schlüssel und statt eckiger Klammern sind es hier geschweifte.

In der zweiten Zeile wird dem Hash `kfz` noch ein Wert hinzugefügt. Dadurch, dass es den Schlüssel `hg` noch nicht gibt, gibt es anschließend ein Element mehr. In der dritten Zeile wird wieder der Schlüssel `hg` verwendet und darum bleibt die Anzahl der Elemente des Hashs gleich. Da hier ein Schlüssel verwendet wird, den es schon gibt, wird nur der Inhalt dieses Elements verändert.

In der vierten Zeile wird ein Skalar `$key` mit dem Inhalt »hg« gefüllt, damit in der letzten Zeile darüber auf das entsprechende Hashelement referenziert werden kann. Wie schon beim Array beobachtet, wird aus dem `%` des Hash ein `$`, sobald `kfz` über den Schlüssel dereferenziert wird und damit nur ein Skalar entnommen wird.

Mit der Funktion `delete` können Einträge aus einer Hashvariable wieder gelöscht werden. Dabei wird der Schlüssel als eindeutige Referenz angegeben. Der Aufruf sieht so aus:

```
delete $kfz('hh');
```

Damit wird der Eintrag mit dem Hamburger Kennzeichen aus dem Hash `kfz` gelöscht.

Interaktiv

Perlskripten nehmen Eingaben von der Standardeingabe durch Zugriff auf <STDIN> entgegen und können mit der Funktion print auf die Standardausgabe ausgeben. Die Aufrufparameter finden sich im Array @ARGV und der Zugriff auf die Umgebungsvariablen erfolgt über die Hashvariable %ENV.

Unterabschnitte

- [Ein- und Ausgabe](#)
 - [Aufrufparameter](#)
 - [Umgebungsvariablen](#)
-

Ein- und Ausgabe

Die Bildschirmausgaben erfolgen über die Funktion `print`. Die Funktion gibt ihre Parameter auf der Standardausgabe aus. `print` hängt keinen Zeilenvorschub an die Ausgabe an. Soll am Ende der Zeile ein Zeilenwechsel erfolgen, muss die Zeichenfolge "n" mit einem Punkt an den Ausgabeparameter gehängt werden. Der Punkt ist der Operator, um zwei Zeichenketten zu verbinden.

Die Eingabe von Zeilen erfolgt durch die Zuweisung der Standardeingabe an einen Skalar.

```
$ask = <STDIN>;
```

Das Programm wartet an dieser Stelle auf die Eingabe des Benutzers, die er mit einem Return abschließen muss. Das Zeilenendezeichen ist damit immer am Ende der Zeichenkette. Da dieses in den seltensten Fällen gebraucht wird, verwendet man die Funktion `chomp`, um die Zeilentrenner aus einer Zeichenkette zu entfernen. Typischerweise sieht eine Eingabe dann so aus:

```
$ask = <STDIN>;  
chomp $ask;
```

« [Interaktiv](#) | [Interaktiv](#) | [Aufrufparameter](#) »

Aufrufparameter

Wie bei Shellskripts finden sich die Kommandozeilenparameter in vordefinierten Variablen. Allerdings unterscheidet Perl zwischen dem Namen des Skripts und seinen Parametern. Der Name des Skripts befindet sich in \$0. Allerdings sind die Aufrufparameter nicht in \$1 und folgende abgelegt, sondern finden sich im vordefinierten Array @ARGV. Der erste Parameter befindet sich in \$ARGV[0]! Das folgende Beispiel zeigt die Kommandozeilenparameter des Skriptes an:

```
[Parameter betrachten]
#!/usr/bin/perl
print "skript name: ", $0, "\n";
print "Parameterzahl: ", $#ARGV, "\n";
print $ARGV[0], "\n";
print $ARGV[1], "\n";
print $ARGV[2], "\n";
```

Die Kombination \$# liefert den höchsten Index des Arrays. Bei einem Parameter ist also \$#ARGV gleich 0! Im Beispiel für den Aufruf des Skripts werden einmal zu viele und einmal gar keine Parameter übergeben.

```
gaston> argv.pl  sonstwas  und dann noch dies
skript name: ./argv.pl
Parameterzahl: 4
sonstwas
und
dann
gaston> argv.pl
skript name: ./argv.pl
Parameterzahl: -1
```

```
gaston>
```

Im letzten Fall entstehen drei Leerzeilen, weil ARGV keinen Inhalt hat. Im Zusammenhang mit for wird später das Skript noch einmal verbessert, damit es auf die variable Anzahl von Parametern reagiert (siehe S. forargv).

Umgebungsvariablen

Für den Zugriff auf die Umgebungsvariablen gibt es die vordefinierte Hashvariable %ENV. Als Schlüssel wird der Name der Umgebungsvariablen verwendet. Dadurch wird der Zugriff durch einfache Zuweisung erreicht. Um den Inhalt der Umgebungsvariablen PRINTER zu ermitteln, verwenden Sie einfach \$ENV{PRINTER}.

```
print $ENV{PRINTER};
```

Um einen neuen Wert für die Umgebungsvariable PRINTER zu setzen, wird ihr einfach ein neuer Wert zugewiesen.

```
$ENV{PRINTER}="laser";
```

Ablaufsteuerung

Wie in anderen Programmiersprachen ist es auch in Perl möglich, bestimmte Programmteile nur unter Bedingungen auszuführen oder zu wiederholen, bis eine Bedingung eintritt. Da die Bedingung immer im Zentrum steht, soll sie zuerst untersucht werden.

Unterabschnitte

- [Bedingungen](#)
 - [if](#)
 - [for](#)
 - [foreach](#)
 - [Sonstige Schleifen: while und until](#)
 - [Funktionen](#)
-

Bedingungen

Eine Bedingung kann wahr oder falsch sein. Sie entsteht durch den Vergleich einer Variablen mit einer anderen Variablen oder einer Konstanten. Solche Konstruktionen nennt man auch boolesche Ausdrücke. Zum Vergleich von Zahlenwerten gibt es die folgenden Vergleichsoperatoren.

[Numerische Vergleichsoperatoren]C|L Operator & Bedeutung

== & gleich

!= & ungleich

< & kleiner

<= & kleiner oder gleich

> & größer

>= & größer oder gleich

<=> & Vergleich. siehe unten

Der Vergleich <=> fällt etwas heraus, da er keinen booleschen Wert zurück gibt. Er liefert 0, wenn beide Werte gleich sind. Er liefert 1, wenn der linke größer als der rechte und -1, wenn der rechte größer als der linke ist.

Für den Vergleich zweier Zeichenketten lauten die Operatoren:

[Vergleichsoperatoren bei Zeichenketten]L|L Operator & Bedeutung

eq & Sind Strings gleich?

ne & Sind Strings ungleich?

lt & Ist erster String kleiner?

le & Ist erster String kleiner oder gleich?

gt & Ist erster String größer?

ge & Ist erster String größer oder gleich?

cmp & Wie strcmp in C: liefert -1 bei kleiner, 0 bei gleich und 1 bei größer

Auch hier liefert der Vergleich cmp keinen booleschen Wert, sondern eine Zahl.

Ob eine bestimmte Zeichenkette in einer Variablen vorkommt, kann ebenfalls als Bedingung verwendet werden. Der Syntax der Suche ist von sed bzw. von vi bekannt.

`$a = /suchmich/i`

Dieser Ausdruck ist wahr, wenn in der Variablen \$a die Zeichenkette »suchmich« vorkommt. Das i hinter dem zweiten Schrägstrich bewirkt, dass bei der Suche nicht zwischen Groß- und Kleinschreibung unterschieden wird. Zwischen den Schrägstrichen können auch reguläre Ausdrücke stehen (siehe S. regexp).

Zum Verknüpfen zweier Vergleiche gibt es die Operatoren and und or. In Anlehnung an die Sprache C können Sie für and auch && und für or auch || schreiben. Der Operator and liefert nur dann einen wahren Wert, wenn beide Ausdrücke wahr sind. Der Operator or wird genau dann wahr, wenn mindestens einer der beiden Ausdrücke wahr ist. Anders ausgedrückt ist or nur dann falsch, wenn

beide Ausdrücke falsch sind.

« [Ablaufsteuerung](#) | **Ablaufsteuerung** | [if](#) »

if

Die Abfrage `if` ermöglicht es, auf Bedingungen zu reagieren. Auf das Schlüsselwort `if` folgt in Klammern die Bedingung. Danach folgen die Befehle, die unter der Bedingung ausgeführt werden sollen in einem Block, der durch geschweifte Klammern eingeschlossen wird.

```
[Einfache Abfrage]
if ( $wert > 5 )
{
    print "Mensch, der wert ist ja über 5!\n";
}
```

Zu dem Befehl `if` gibt es auch ein `else`, dem wiederum ein Block von Anweisungen angehängt werden kann. Dieser Block wird dann ausgeführt, wenn die Bedingung falsch ist.

Mit dem Befehl `elsif` ist es möglich, auf eine weitere Bedingung abzufragen. So können Sie beispielsweise einen Block reagieren lassen, wenn die Variable 3 ist, einen anderen, wenn sie 5 ist und mit einem `else` alle übrigen Werte abhandeln. Das folgende Beispiel zeigt ein `if` mit `elsif` und `else`. Hier werden Zeichenketten verglichen.

```
[Begrüßung]
#!/usr/bin/perl -w
$ask = <STDIN>;
chomp($ask); # wirf den Zeilenvorschub weg!
if ( $ask eq "arnold" )
{
    print "Hallo, Arnold!\n";
}
elsif ( $ask eq "willemer" )
{
    print "Hochverehrter Herr willemer!\n";
}
else
{
    print "Was willst Du denn hier, $ask?\n";
}
```

Zunächst wird eine Zeichenkette von der Tastatur eingelesen. Da das erst mit der Eingabe der Taste Return beendet ist, befindet sich der Zeilenvorschub am Ende der Zeichenkette in der Variablen `$ask`. Die Funktion `chomp()` eliminiert den Zeilenvorschub.

Es folgt die eigentliche Abfrage `if`. Zunächst wird geprüft, ob die Eingabe »arnold« lautete. Dann wird Arnold mit Hallo begrüßt. Das `elsif` heißt soviel wie »andernfalls wenn«. Hier wird geprüft, ob »willemer« eingegeben wurde. Dann fällt die Begrüßung gleich sehr viel formeller aus. Das `else` behandelt alle anderen Fälle.

for

Die Schleife `for` hat ihren Einsatz, wenn abzählbare Durchläufe einer Schleife gebraucht wird. Perl kennt zwei Formen des `for`. Da ist einmal die Zählschleife, die hier behandelt wird, und dann die Schleife `foreach`, mit der sich der nächste Abschnitt befasst.

```
[Zählen mit for]
for ($i=0 ; $i<10 ; $i++ )
{
    print "$i\n";
}
```

In der Klammer hinter dem `for` sind drei Anweisungen, die je durch ein Semikolon getrennt werden. Als erstes erscheint die Startanweisung. Sie wird einmal vor Beginn der Schleife ausgeführt. In diesem Fall setzt sie den Inhalt des Skalars `$i` auf 0. Das zweite ist die Schleifenbedingung. Sie wird vor jedem Durchlauf der Schleife überprüft. Trifft sie nicht mehr zu, wird die Schleife beendet. Der dritte Teil wird nach jedem Durchlauf am Ende des Schleifenblocks durchgeführt. Im Beispiel steht hier die Anweisung, den Skalar hochzuzählen.

Das folgende Beispiel zeigt alle Aufrufparameter des Perlskripts an.

```
[Parameteraufzählung]
#!/usr/bin/perl
print "skript name: ", $0, "\n";
print "Parameterzahl: ", $#ARGV, "\n";
print "Parameterliste: \n";
for ($i=0; $i<=$#ARGV; $i++)
{
    print "$i.: ",
        $ARGV[$i], "\n";
}
\}$
[tex2html_wrap_inline14220]
```

Die Kombination `$#` liefert den höchsten Index des Arrays. Ist das Array leer, ist dieser Wert -1. Bei einem Parameter ist `$#ARGV` gleich 0! Aus diesem Grund läuft `$i` in der Schleife auch bis `<= $#ARGV`. Für den C-Programmierer etwas gewöhnungsbedürftig ist, dass `$i` auch innerhalb des Strings ausgewertet wird. Ein Beispiel für den Aufruf des Skripts:

```
gaston> argv.pl  sonstwas  und dann noch dies
skript name: ./argv.pl
Parameterzahl: 4
Parameterliste:
0.: sonstwas
1.: und
2.: dann
3.: noch
4.: dies
gaston> argv.pl
skript name: ./argv.pl
Parameterzahl: -1
Parameterliste:
gaston>
```

foreach

Die Schleife `foreach` ist ein Spezialfall der `for`-Schleife, die Werte einer Liste durchläuft. Sie ist vergleichbar mit der `for`-Schleife in Shellskripten. Dem Schlüsselwort `foreach` folgt als erstes ein Skalar, das nacheinander alle Werte der darauffolgenden Liste annimmt.

```
[foreach über ein Array]
foreach $i (@array) {
    print $i."n";
}
```

Hier nimmt der Skalar `$i` nacheinander die Werte des Arrays `@array` an. In jedem Durchlauf wird also ein Element von `@array` angezeigt. Das folgende Beispiel wertet wieder die Aufrufparameter aus. Hier sehen Sie, wie `foreach` die Schleife erheblich vereinfacht.

```
[Parameteraufzählung vereinfacht]
#!/usr/bin/perl
print "Parameterliste: n";
for ($i (@ARGV)
{
    print "$i n";
}
```

Nun soll eine Hashvariable ausgewertet werden. Sie soll in der alphabetischen Reihenfolge ihrer Schlüssel angezeigt werden.

```
[foreach über Hashvariablen]
%kfz = ('sl', 'Schleswig', 'fl', 'Flensburg', 'hh', 'Hamburg');
$kfz{'hg'} = 'Bad Homburg';
foreach $key (sort keys(%kfz)) {
    print "Key = $key, Value = $kfz{$key}n";
}
```

Die Schleifenvariable `$key` nimmt nacheinander die Werte des dahinter stehenden Ausdrucks an. Im Zentrum dieses Ausdrucks steht die Hashvariable `%kfz`. Die Funktion `keys` liefert ein Array mit allen Schlüsselwerten der Hashvariablen. Auf diese wird dann die Funktion `sort` angewendet, so dass `$key` nacheinander die Schlüssel in sortierter Reihenfolge annimmt.

Statt dem Schlüsselwort `foreach` kann auch `for` verwendet werden. Allerdings erhöht die Verwendung von `foreach` an solchen Stellen die Lesbarkeit.

Sonstige Schleifen: while und until

Perl kennt für Schleifen die Schlüsselworte `while` und `until`. Beim `while` bleibt das Programm innerhalb der Schleife, sofern die Bedingung erfüllt ist. Bei `until` wird die Schleife verlassen, sobald die Bedingung erfüllt ist. Das eine ist also die Negation des anderen. Beide Schlüsselworte können zu Anfang oder zu Ende der Schleife stehen, je nachdem, wo die Bedingung abgefragt werden soll. Steht die Bedingung am Ende, wird die Schleife in jedem Fall einmal durchlaufen. In diesem Fall eröffnet das Schlüsselwort `do` die Schleife.

```
[while am Schleifenanfang]
$i = 0;
while ($i<3) {
    print "$i\n";
    $i++;
}
```

Die Bedingung wird zu Beginn geprüft. Träfe sie zu Beginn nicht zu, würde die Schleife nicht durchlaufen.

```
[while am Schleifenende]
$i = 0;
do {
    print "$i\n";
    $i++;
} while ($i<3);
```

Da hier die Bedingung am Ende geprüft wird, würde die Schleife einmal durchlaufen, auch wenn die Bedingung bereits zu Anfang nicht zuträfe.

```
[until am Schleifenanfang]
$i = 0;
until ($i>=3) {
    print "$i\n";
    $i++;
}
```

Im Gegensatz zur ersten `while`-Schleife wird hier die negierte Abfrage geprüft. Im Grunde ist das die gleiche Schleife wie die erste, die mit `while` erstellt wurde. Die Negierung einer Bedingung ist im Allgemeinen aber schlechter lesbar als die Verwendung von `until` statt `while`.

```
[until am Schleifenende]
$i = 0;
do {
    print "$i\n";
    $i++;
} until ($i>=3);
```

Zu guter Letzt noch die vierte mögliche Variante. Hier wird ebenfalls die Schleife mindestens einmal durchlaufen, bevor die Bedingung geprüft wird.

Die folgende Schleife ist ein Beispiel für das Aufspalten eines Arrays in einzelne Skalare. Die

Bedingung ist lediglich das Array. Hier wird genutzt, dass die Umformung eines Arrays in ein Skalar die Anzahl der Elemente ergibt. Sobald das Array also leer ist, wird es 0 zurückgegeben. Diese 0 wird von `while` als falsch interpretiert und führt zum Ende der Schleife.

[Paarweises Abführen]

```
while (@array) {
    ($a, $b, @neu) = @array;
    print $a." ".$b."n";
    @array = @neu;
}
```

Im Innern der Schleife wird das Array durch die Zuweisung jeweils in zwei Skalare und ein neues Array aufgespalten. Die Skalare werden paarweise ausgegeben und das Restarray dem Original zugewiesen.

Im Zusammenhang mit CGI-Skripten sind immer wieder Strings wie der folgende auszuwerten. Sie enthalten den Inhalt einer Eingabemaske. Ziel ist es, die einzelnen Eingaben zu trennen. Die einzelnen Eingaben sind als Zuweisungen dargestellt, also links Variable, dann ein Gleichheitszeichen und dann der Wert. Die Zuweisungen jeweils sind durch ein `&` getrennt.

Name=willemer&Adresse=Ihre+Adresse%0D%0Aort&Anrede=Frau

Das Skript soll die Zeichenkette aufbrechen und in einem Hash ablegen und den Hash ausgeben. Statt der Ausgabe würde man in der Praxis die Werte in eine Datenbank stellen oder zu einer Datenbankabfrage umformen.

[CGI-Zerlegung]

```
#!/usr/bin/perl -w
use strict;
my %input;
my $zeile; my $key; my $wert;
my @zuweisungen; my @neu;
```

```
my $input = "Name=Otto&Adresse=Mein+Weg%0D%0Aort&Anrede=Frau";
@zuweisungen = split("&", $input);
while (@zuweisungen) {
    ($zeile, @neu) = @zuweisungen;
    ($key, $wert) = split("=", $zeile);
    $wert = s/+//g;          # + durch leer ersetzen!
    $input{$key} = $wert;
    @zuweisungen = @neu;
}
# Ausgabe
foreach $key(sort keys(%input)) {
    print "key = $key, value = $input{$key}n";
}
```

« [foreach](#) | [Ablaufsteuerung](#) | [Funktionen](#) »

Funktionen

Eine Funktion wird mit dem Schlüsselwort `sub` definiert. Es folgt der Name der Funktion und in geschweiften Klammern der Block von Befehlen, der ausgeführt wird, wenn die Funktion aufgerufen wird. Eine Funktion kann von einer beliebigen Stelle durch ihren Funktionsnamen aufgerufen werden. Nach Ausführung der Befehle in der Funktion kehrt der Programmablauf zu dem Befehl zurück, der dem Funktionsaufruf folgt.

```
[Simple Funktion]
sub linie {
    print '-' x 79 . "\n";
}

...
linie;

...
linie;
```

Funktionen haben den Vorteil, dass Programmzeilen nicht an verschiedenen Stellen des Programms wiederholt werden müssen. Das Vermeiden von Wiederholungen mindert Fehlerquellen. Fehlerkorrekturen in Funktionen, die häufiger aufgerufen werden, lassen mehrere Stellen des Programms davon profitieren. Daneben sorgen Funktionen für Übersicht. Da Details in den Funktionen verborgen werden, entsteht ein Blick auf die Struktur des Gesamtprogramms.

An Funktionen können auch Parameter übergeben werden. Beim Aufruf werden sie einfach hinter dem Funktionsnamen aufgezählt. Vom inneren der Funktion können Sie sie "über das Array `@_`" zugreifen. Das Beispiel `linie` ist so erweitert worden, dass als Parameter übergeben wird, wieviele Striche für die Linie verwendet werden sollen.

```
[Funktion mit Parameter]
sub linie {
    print '-' x $_[0] . "\n";
}

...
linie(5);

...
linie 15;
```

Es können lokale Variablen mit Hilfe der Schlüsselworte `local` oder `my` innerhalb der Funktion definiert werden. Mit `my` definierte Variablen sind nur innerhalb der Funktion bekannt. Eine mit `local` definierte Variable kann auch von einer Funktion zugegriffen werden, die von der aktuellen Funktion aufgerufen wird.

Dateien

Der Umgang mit Dateien besteht einmal aus dem Lesen und Schreiben von Dateien. Darüber hinaus ist es für eine Skriptsprache auch wichtig, dass man Dateien löschen oder umbenennen kann.

Unterabschnitte

- [Schreiben und Lesen](#)
 - [Umgang mit Dateien](#)
-

Schreiben und Lesen

Perl ist eine ideale Sprache, um Texte auszuwerten. Um das nutzen zu können, muss man auf die Dateien, in denen die Texte stehen, zugreifen können. Im ersten Schritt wird eine Datei eröffnet. Dazu dient die Funktion `open`. Der erste Parameter ist das Dateihandle (Handle engl. Handgriff). Ein Handle ist eine Kennung für eine Ressource. Daran erkennt das Betriebssystem, welche Datei gemeint ist. Das Handle wird für alle weiteren Zugriffe auf die Datei gebraucht. Der zweite Parameter ist der Name der Datei, die geöffnet werden soll. Im folgenden Beispiel wird eine Datei geöffnet und gleich wieder geschlossen.

```
open(HANDLE, $filename) || die "Datei nicht zugreifbar";
close(HANDLE);
```

Das Oderzeichen hinter dem Dateinamen bewirkt, dass bei einem Fehlschlag der Funktion `open()` die dahinterliegende Befehlsfolge abgearbeitet wird. Hier besteht sie aus der Funktion `die()` (engl. sterben), die das Programm beendet und die als Parameter übergebene Zeichenkette als letzten Gruss auf der Konsole ausgibt.

Aus einer Datei liest man mit Hilfe des beim `open()` ermittelten `HANDLE` auf die gleiche Weise, wie schon weiter oben von der Standardeingabe gelesen wurde.

```
$zeile = <HANDLE>;
```

Die Ausgabe in eine Datei erfolgt analog über die Funktion `print()`. Als erster Parameter wird ihr das Dateihandle mitgegeben.

```
print HANDLE $value;
```

Das folgende Skript ist eine Implementation des Kommandos `cat` in Perl. Wie der Originalbefehl gibt das Skript `cat.pl` den Inhalt aller ihm als Parameter übergebenen Dateien auf der Standardausgabe aus.

```
[Implementierung von cat in Perl]
#!/usr/bin/perl -w
use strict;
my $datei;    # der jeweilige Dateiname
my $zeile;    # die Zeile, die aus der Datei gelesen wird.

foreach $datei (@ARGV)
{
    open(HANDLE, $datei) || die "Fehler beim Öffnen $datei";
    while ($zeile = <HANDLE>)
    {
        print $zeile;
    }
    close HANDLE;
}
```

Die äußere `foreach`-Schleife durchläuft das Parameterarray `@ARGV`. Jeder der Parameter wird mit

`open` geöffnet. Dann durchläuft das Programm die innere Schleife, in der jede Zeile in den Skalar `$zeile` eingelesen wird und dann per `print` auf die Standardausgabe gegeben wird. Die Schleife bricht ab, wenn die Variable `zeile` nicht mehr zu füllen ist. Nach Ende der Schleife wird die Datei geschlossen und die `foreach`-Schleife übernimmt den nächsten Parameter.

« [Dateien](#) | [Dateien](#) | [Umgang mit Dateien](#) »

Umgang mit Dateien

Perl verfügt über einige Funktionen zur Behandlung von Dateien. Diese sind angelehnt an die UNIX Systemaufrufe. Dazu gehören:

[Dateibefehle]L|L Befehl & Wirkung

unlink \$filename & Löschen einer Datei

rename \$filename,\$neuname & Ändern des Namens einer Datei

mkdir \$filename & Erzeugen eines Verzeichnisses

rmdir \$filename & Löschen eines Verzeichnisses

Weitere Funktionen finden Sie auf der Manpage: man perlfunc.

Wie bei UNIX-Skripten können Dateien auf Existenz und andere Eigenschaften geprüft werden. Beispielsweise wird mit -f getestet, ob der nachfolgende Name, hier .rhosts, eine Datei bezeichnet.

[Dateilöschen mit Vorabprüfung]

```
if ( -f '.rhosts' ) {  
    unlink '.rhosts';  
}
```

Die wichtigsten unären Operatoren sind:

[Dateieigenschaften prüfen]L|L Flag & Bedeutung

-r -w -x & vom effektiven UID/GID les-, schreib bzw. ausführbar

-R -W -X & vom realen UID/GID les-, schreib bzw. ausführbar

-f & existiert als Datei

-d & existiert als Verzeichnis

-l & existiert als symbolischer Link

« [Schreiben und Lesen](#) | **Dateien** | [Perl und UNIX](#) »

Perl und UNIX

Perl entstand unter UNIX. So nutzt Perl viele Möglichkeiten, die UNIX ihm bietet. Inzwischen ist Perl auf beinahe jeder Plattform verfügbar. Wer die Möglichkeit nutzen will, seine Perlskripte auch auf anderen Systemen zu verwenden, sollte Spezifika von UNIX vermeiden. Aber für den Systemadministrator, der sich ein paar Hilfsroutinen schreiben will, ist die Nähe von Perl zu UNIX ungeheuer praktisch. Dass solche Skripten dann nicht portabel sind, wird ihn weniger interessieren.

Unterabschnitte

- [Aufruf von UNIX-Programmen](#)
 - [UNIX Systemprogrammierung](#)
-

Aufruf von UNIX-Programmen

Perl kennt auch die Verwendung von backquotes. Damit können UNIX Programme aufgerufen werden und ihre Ausgabe im Programm weiterverarbeitet werden. Ein einfaches Beispiel ist die Ermittlung des aktuellen Datums:

```
$datum = `date +%D`;  
print "$datum";
```

Der Befehl `date` wird aufgerufen und dessen Ausgabe an die Standardausgabe wird in den Skalar `$datum` umgeleitet. Die einschließenden Striche sind keine Apostrophe, sondern das Zeichen, das dieselbe Richtung hat wie der Backslash. Aus diesem Grund wird das Zeichen auch Backquote genannt. Die Funktion `print` liefert das Ergebnis auf dem Bildschirm.

Das Gleiche funktioniert auch, wenn die Ausgabe des Befehls mehrere Zeilen umfasst. Sie müssen dessen Ausgabe allerdings in einem Array speichern. Als Beispiel wird hier der Befehl `ls` verwendet.

```
@dir = `ls`;
```

[« Perl und UNIX](#) | [Perl und UNIX](#) | [UNIX Systemprogrammierung](#) [»](#)

UNIX Systemprogrammierung

Unter Perl stehen die meisten UNIX Systemaufrufe, die später im Buch behandelt werden, zur Verfügung. Selbst Aufrufe wie `fork()` und `kill()` sind möglich. Die Verwendung von Sockets ist durchaus gängige Praxis. Da auch Netzwerkfunktionen wie `accept()` oder `connect()` verfügbar sind, ist es möglich, Netzwerkserver und Clients zu programmieren.

Grafische Oberfläche: Tk

Dieser Ausflug in die Welt der grafischen Programme kann natürlich das Thema nicht völlig erschöpfen. Es macht aber einfach Spaß, wenn man sieht, wie schnell man mit Perl grafische Oberflächen erstellen kann.

Um mit Perl X-Clients zu schreiben, bedienen Sie sich des Moduls Tk. An sich wurde Tk für die Skriptsprache Tcl entwickelt. Aber auch mit Perl können Sie Tk benutzen. Das folgende Beispiel erstellt ein Fenster mit einem Label, also einer Anzeige für Text, in der das Wort »Huhu« steht. Darunter erscheint ein Button mit der Aufschrift »Schluss«, der das Programm beendet.



Im ersten Schritt wird das Hauptfenster generiert. Dabei wird die Methode `new` der externen Klasse `MainWindow` verwendet um ein Fenster zu erzeugen, das anschließend über die Objektvariable `$meinFenster` erreichbar ist. Falls sich das für Sie sehr nach objektorientiertem Vokabular anhört, liegen Sie richtig. Mit Perl können Sie durchaus objektorientierte Programmierung betreiben. Dann wird nacheinander die Methode `Label` und `Button` aufgerufen, die ein entsprechendes Element generieren.

```
#!/usr/bin/perl -w
use Tk;

my
  $meinFenster = MainWindow->new;$
  [tex2html_wrap_inline16001]meinFenster->Label( -text=>"Huhu" )->pack;

  $meinFenster->Button(-text => "Schluss",
                      -command => [$
  [tex2html_wrap_inline16003]meinFenster => 'destroy']
                      )->pack;
MainLoop;
```

Gerade beim Button ist schön zu sehen, dass bei der Erstellung bereits alle relevanten Eigenschaften eines Buttons erzeugt werden, die Beschriftung (`text`) und die aufzurufende Funktion bei Betätigung des Buttons. Letzteres führt dazu, dass das Fenster zerstört und damit das Programm beendet wird. Zu guter Letzt läuft das Programm in die allen grafischen Oberflächen typische Endlosschleife, hier `MainLoop`. Das bedeutet, dass das Programm nicht mehr von sich aus agiert, sondern auf die Benutzeraktivitäten wartet und dann mit den hinterlegten Rückruffunktionen reagiert.

Sie finden im Listing zwei ungewöhnliche Pfeile, die im bisherigen Kapitel nicht behandelt wurden. Der Pfeil, der sich aus Bindestrich und Größerzeichen zusammensetzt, kann man sich als Zeiger auf Bestandteile grafischer Objekte vorstellen. So kann ein Fenster Funktionen aufrufen, die sich auf das Fenster beziehen oder Sie können die zum Fenster gehörigen Variablen auslesen.

Der zweite Pfeil, der aus einem Gleichheits- und einem Größerzeichen besteht, ist einfacher zu erläutern. An sich kann man ihn sich als Ersatz für ein Komma vorstellen. Sie können ihn auch durch ein Komma ersetzen. Der Pfeil hat den Vorteil, Parameterpaare deutlicher hervorzuheben. So bezieht sich das »Huhu« auf die Option -text.

Die Funktion pack muss auf alle Elemente des Fensters angewandt werden, damit die Elemente im Fenster angeordnet und damit sichtbar werden. Solange nur mit wenigen Elementen gearbeitet wird, reicht diese Erklärung. Auf das Thema und die verschiedenen Anordnungsmöglichkeiten wird ab Seite pack genauer eingegangen.

Unterabschnitte

- [Widgets und Ressourcen](#)
- [Kontrollelemente](#)
 - [Label](#)
 - [Button](#)
 - [Listbox](#)
 - [Scrollbars](#)
 - [Scale](#)
 - [Entry](#)
 - [Menüs](#)
- [Widgetanordnung](#)
 - [Informationen](#)

Widgets und Ressourcen

Die Kontrollelemente unter X gehören zu den Widgets. Widgets sind eigenständige Bestandteile der grafischen Oberfläche. Neben den Kontrollelementen gibt es Containerwidgets, die andere Widgets aufnehmen und diese ausrichten. Wie schon im Kapitel X Window System erläutert wurde, haben Widgets Eigenschaften, die über ihre Namen zu beeinflussen sind. Diese Eigenschaften nennt man Ressourcen.

[Allgemeine Ressourcen]L|L|L Ressource & Bedeutung & Werte

-background -bg & Hintergrundfarbe &Farbangabe

-foreground -fg & Vordergrundfarbe &Farbangabe

-relief & 3D-Effekt & raised, sunken, flat, ridge (umrahmt)

-borderwidth -bd & Randstärke & numerischer Wert

-anchor & Ausrichtung & 'n', 'w', 's', 'e' oder 'center'

Sie können Ressourcen nachträglich ändern, indem Sie die Funktion `configure` des jeweiligen Widgets mit der Ressource als Parameter aufrufen. Beispielsweise können Sie den Text eines Labels folgendermaßen ändern:

```
my [tex2html_wrap_inline16005]mw->Label( -text=>'Beschriftung' );
```

```
...
```

```
$anzeige->configure(-text=>'Anderer Text');$  
[tex2html_wrap_inline16007]
```

Weitere Informationen gibt es unter `man Tk::options`.

Unterabschnitte

- [Label](#)
 - [Button](#)
 - [Listbox](#)
 - [Scrollbars](#)
 - [Scale](#)
 - [Entry](#)
 - [Menüs](#)
-

Kontrollelemente

Für alle interaktiven Programme sind die Kontrollelemente zentraler Bestandteil ihrer Fenster. Die wichtigsten Elemente, ihre Ressourcen und der Umgang mit ihnen wird in diesem Abschnitt dargestellt.

Label

Ein Label ist ein Beschriftungsfeld. Sie können es mit einem Text und einem Bild füllen. Davon abgesehen, ist es sehr temperamentslos. Es reagiert auf keine Ereignisse. In den allermeisten Fällen dient es nur der Beschriftung.

[Label Ressourcen]L|L|L Ressource & Bedeutung & Werte
-text & Beschriftung & Zeichenkette
-font & Schrift & Zeichensatzbezeichnung

Eine Sonderform des Labels ist die Message. Man kann ihr längere Texte angeben, die sie je nach Raum selbstständig umbricht.

Button

Der einfache Button ist ein Knopf, der auf das Anklicken mit der linken Maustaste reagiert. Ansonsten kann man ihn wie ein Label mit einem Text oder einem Bild versehen.

```
$meinFenster->Button(-text => "Schluss",  
                    -command => sub{faerbe('red')});
```

Neben der Beschriftung ist die wichtigste Eigenschaften eines Buttons, dass man ihn anklicken kann. Mit der Option -command wird festgelegt, welche Aktion dann ausgelöst wird. Oben wird bei Druck auf den Button die Funktion faerbe aufgerufen. Man nennt eine solche Funktion auch Callback. Solche Callbacks sind die Schnittstelle zwischen den Benutzeraktionen an der Oberfläche und dem eigentlichen Programm.

Der Checkbutton ist eine Sonderform des Buttons. Durch Anklicken erhält er eine Marke. Wenn Sie ihn

ein weiteres Mal anklicken, geht der Button in seinen Ursprungszustand zurück.

```
my $schoen=  
  $mw->Checkbutton(-text =>"schön",  
                   -anchor=>'w')->pack(-fill,'x');  
my $stark=  
  $mw->Checkbutton(-text =>"stark",  
                   -anchor=>'w')->pack(-fill,'x');  
my $klug=  
  $mw->Checkbutton(-text =>"klug",  
                   -anchor=>'w')->pack(-fill,'x');$
```



Die Ressource `-anchor` bewirkt eine Ausrichtung nach links, wenn ihr Wert wie oben `'w'` ist. Das wirkt aber nur dann, wenn bei der Funktion `pack` angegeben ist, dass das Widget den gesamten Raum in X-Richtung füllen soll. Auf die Ausrichtung und Anordnung von Widgets wird später näher eingegangen (siehe S. `pack`).

Der Radiobutton ist eine andere Variante der Buttons. Man könnte ihn als eine Weiterentwicklung des Checkbutton ansprechen. Mehrere Buttons werden zusammengefasst. Es darf nur einer angewählt sein. Er hat seinen Namen von den Stationstasten eines Radios, von denen ja auch immer nur eine gleichzeitig angewählt sein kann. Wird eine andere gedrückt, springt die bisher gedrückte Taste heraus.



```
my $mw = Mainwindow->new;  
my  
  $radvar='red';  
\par$  
-->
```

```

mw->Radiobutton(-text => 'rot',
                -variable=>
                $radvar,
                -value=>'red',
                -anchor=>'w')->pack(-fill,'x');$
-->
mw->Radiobutton(-text => 'gelb',
                -variable=>
                $radvar,
                -value=>'yellow',
                -anchor=>'w')->pack(-fill,'x');
MainLoop;$
-->

```

Relevant für die Funktionalität sind die Optionen `-variable` und `-value`. In der Variablen `$radvar` legen die Radiobuttons nicht nur ihren Wert ab, sie stellen anhand der Variablen auch fest, ob sie ausgewählt sind oder nicht. Dazu hat jeder Radiobutton einen eigenen Wert, die hinter in der Ressource `-value` abgelegt ist.

Listbox

Eine Listbox kann mehrere Zeilen aufnehmen. Zunächst wird sie generiert wie andere Kontrollelemente. Eine wichtige Ressource ist die Höhe, die angibt, wieviele Zeilen sichtbar sind. Die Listbox kennt die Kommandos `insert` zum Einfügen von Zeilen, `delete` zum Löschen und `get`, um den Inhalt einer Zeile auszulesen.

```

#!/usr/bin/perl -w
use strict;
use Tk;

my $mw = MainWindow->new;
my mw->Listbox(-height=> 5);
list->insert(0,'gelb','blau','grün','rot','schwarz','weiß');
MainLoop;

```

[Listbox Ressourcen]L|L Ressource & Werte
`-height` & Anzahl der sichtbaren Zeilen

Scrollbars

Scrollbars werden im Zusammenhang mit anderen Widgets gebraucht, wenn der Raum im Fenster zu klein ist, um das Widget komplett darzustellen. Dann kann man einen Scrollbar verwenden, um den Ausschnitt auszuwählen, den man sehen möchte. So braucht der Scrollbar die Information, welches Widget er kontrollieren soll. Auf der anderen Seite braucht das kontrollierte Widget auch Informationen darüber, dass es von einem Scrollbar kontrolliert wird. Man löst das Problem, indem man zunächst das zu kontrollierende Widget erzeugt, dann den Scrollbar und schließlich dem Widget durch Ressourcenänderung mitteilt, dass es einen Scrollbar besitzt. Im folgenden Beispiel werden diese Schritte an einer Listbox demonstriert.

```

#!/usr/bin/perl -w
use strict;
use Tk;

```

```

my
  $mw = Mainwindow->new;
# Erzeuge die Listbox
my$

list =
  $mw->Listbox(-height=> 5)->pack(-side, 'left');
# Dann den passenden Scrollbar
my$

scroll = MATH
  $mw->Scrollbar(-command, [yview=>$list]);

  $scroll->pack(-side,'right', -fill,'y');
# Nun existiert ein Scrollbar und die Listbox sollte das wissen$

list->configure(-yscrollcommand => ['set',
  $scroll]);
# Fuelle werte in die Listbox...$
list->insert(0,'gelb','blau','grün','rot','schwarz','weiß');
MainLoop;

```



Da Scrollbars fast immer eingesetzt werden, um andere Widgets zu steuern, gibt es die Sonderform Scrolled. Damit werden kontrolliertes Widget und Scrollbar in einem Schritt erzeugt.

```

#!/usr/bin/perl -w
# use strict;
use Tk;

my
  $mw = Mainwindow->new;
my$
  -->
scrlist =
  $mw->Scrolled(Listbox,-height,5,-scrollbars=>'e');$
  -->
scrlist->insert(0,'gelb','blau','grün','rot','schwarz','weiß');

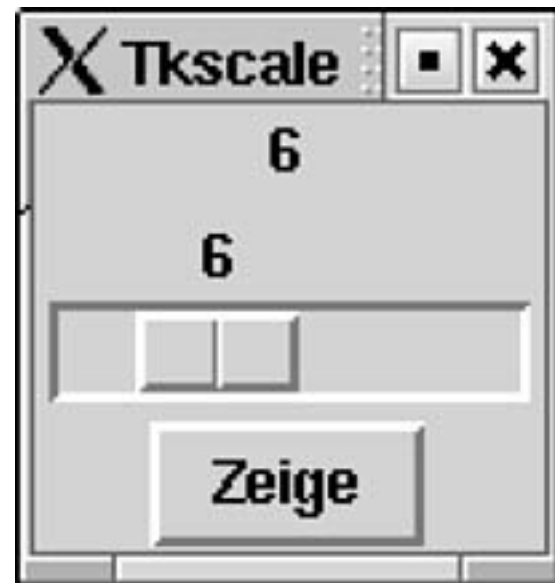
  $scrlist->pack;
MainLoop;$
  -->

```

Hinter der Option `-scrollbars` steht in der Zeichenkette, an welcher Seite des Widgets der Scrollbar angebracht sein soll. Die Seite wird durch die englischen Kürzel der Himmelsrichtungen (n, e, s, w) angegeben. Im Beispiel oben ist der Scrollbar also an der rechten Seite. Durch Angabe der Zeichenkette 'se' wird sowohl rechts als auch unten ein Scrollbar angefügt.

Scale

Das Widget Scale ist ein Schieber, mit dem Sie Zahlenwerte einstellen können. Mit dem Kommando `get` lässt sich der Zahlenwert auslesen. Das folgende Beispiel erzeugt einen solchen Schieber und übernimmt auf Drücken des Buttons den Wert in ein Label.



```
#!/usr/bin/perl -w
use strict;
use Tk;

my $mw = MainWindow->new;
my mw->Label( -text=>"" )->pack;
my mw->Scale(-from=>1, -to=>20,
            -orient=>'horizontal')->pack;

$mw->Button(-text => "Zeige", -command => sub\{zeige()\},)->pack;
MainLoop;
\par sub zeige \{
    my$
    -->
    val = anzeige->configure(-text=>
```

Das Widget Scale verwendet folgende Ressourcen:

[Scale Ressourcen]L|L Ressource & Werte
-label & Beschriftung
-from & Min. Wert (numerisch)
-to & Max. Wert (numerisch)
-length & Zahlenwert
-orientation & horizontal oder vertical

Entry

Das Widget Entry dient der Eingabe einer Zeichenkette. Das folgende Beispiel erzeugt ein Fenster, mit einem Eingabefeld, einem Label und einem Button. Wenn Sie etwas in das Eingabefeld eintippen und dann den Button anklicken, wird der Inhalt im Label angezeigt.



```
#!/usr/bin/perl -w
use strict;
use Tk;

my $mw = MainWindow->new;
my mw->Label( -text=>"" )->pack;
my mw->Entry()->pack;

$mw->Button(-text => "Zeige", -command => sub\{zeige()\},)->pack;
MainLoop;
\par sub zeige \{
    my$
    -->
    val = anzeige->configure(-text=>
```

[Entry Ressourcen]L|L Ressource & Werte

-label & Beschriftung

-textvariable & Variable

Neben diesen Grundfunktionen gibt es natürlich eine Menge Möglichkeiten, mit Eingabefeldern umzugehen. Dabei ist allein der Umgang mit Selektionen, also Markierungen im Text sehr umfangreich. Genauere Informationen bekommt man über `Tk::Entry`.

Menüs

Zu jedem etwas größeren Programm gehört auch ein Menübaum. Dieser besteht aus mehreren Elementen, die im Programm nacheinander generiert werden. Zunächst wird die Menüleiste (engl. menubar) erzeugt. Sie befindet sich immer am Kopf eines Fensters. Diese Position nennt sich unter Tk `toplevel`. In der Menüleiste werden kaskadierende Buttons eingesetzt, die beim Anklicken das eigentliche Menü aufklappen. Dahinter endlich verbergen sich die eigentlichen Menüpunkte, die bei Tk `Command` heißen und wie gewöhnliche Buttons funktionieren. Auch ihnen wird bei Erzeugung eine

Callbackfunktion zugeordnet. Allerdings ist das nicht zwingend. Im Beispiel ist als zweite Kaskade eine Farbenselektion eingebaut, hinter der sich Radiobuttons verbergen. Wegen des Unterhaltungswertes sind die Menüpunkte auch gleich in der Farbe dargestellt, die man damit anwählen kann.



```
#!/usr/bin/perl -w
use strict;
use Tk;

my $mw = MainWindow->new;
my mw->toplevel;

# Die Menueleiste wird in den Kopf des Fensters gehaengt
my toplevel->Menu(-type => 'menubar');

$toplevel->configure(-menu => $
-->
menubar);

# Nun bauen wir ein Datei-Menue
my menubar->cascade(-label => 'Datei',
                    -tearoff => 0);

$datei->command(-label => 'Zeige', -command => sub\{zeige()\});$

datei->command(-label => 'Piep', -command => datei->command(-label => 'Quit', -command =>
$mw=>'destroy']));
my$

farbe =
$menubar->cascade(-label => '~Farbe',
                -tearoff => 0);

\par my$

farbe->radiobutton(-label => 'rot',
                  -command => sub\{faerbe('red')\},
                  -background => 'red');

$farbe->radiobutton(-label => 'gelb',
```

```

        -command => sub\{faerbe('yellow')\},
        -background => 'yellow');
\par my$

anzeige =
    $mw->Label( -text=>"" )->pack;
my$

entry = MATH
    $mw->Entry()->pack;$

mw->Button(-text => "Zeige",
           -command => sub{zeige()},
           )->pack;
MainLoop;

```

```

sub zeige {
    my entry->get;

    $anzeige->configure(-text=>$
val);
}

```

```

sub faerbe {

    $entry->configure(-background =>$_[0]);
}

```

Inhaltlich ist das Programm nichts anderes als das Beispielprogramm für das Eingabefeld Entry. Allerdings gibt es nun die Möglichkeit, aus dem Menü heraus die Zeigefunktion zu aktivieren.

« [Widgets und Ressourcen](#) | [Grafische Oberfläche: Tk](#) | [Widgetanordnung](#) »

Unterabschnitte

- [Informationen](#)

Widgetanordnung

Im vorigen Abschnitt wurde bereits von der Funktion `pack` ausgiebig Gebrauch gemacht. Die einfachste Variante ist der Aufruf ohne Parameter. Dann werden die Widgets nacheinander in das Fenster gestopft und ihre Anordnung ist eher zufällig. Immerhin erfüllen die Widgets auch so ihre Funktionalität.

Durch die Option `-side` kann die Richtung angegeben werden, von der aus das Fenster aufgefüllt wird. Die üblichen Argumente der Option sind `'top'` und `'left'`. Dann werden die Widgets nacheinander von oben nach unten, respektive von links nach rechts aufgefüllt. Natürlich gibt es auch die Argumente `'bottom'` und `'right'`. Der Vorgabewert ist übrigens `'top'`. Das Vorgehen kann man sich vorstellen, als würde man mit der Kreissäge ein Stück des Fensters abschneiden. Ein Aufruf von `pack` mit der Option `-side 'top'` würde also für das Widget die obere Kante des Fensters abtrennen. Es verbleibt zur freien Verteilung der untere Rest. Diesen können Sie wiederum an einer der Kanten absägen. Ein Problem ergibt sich, wenn Sie folgendes Fenster aus Label, Listbox, Scrollbar und zwei Buttons erstellen wollen.



Das Absägen des Titels ist noch trivial. In dem Moment, wo Sie aber links die Listbox abtrennen, bekommen Sie den Button nicht mehr unter die Listbox. Schneiden Sie dagegen die Listbox oben ab, fehlt Ihnen der Scrollbar an der rechten Seite. Man müsste Listbox und Scrollbar gemeinsam abtrennen können. Genau zu diesem Zweck gibt es Rahmen, die Tk Frame nennt. Sie werden wie ein normales Widget erzeugt, bekommen aber ihre Bedeutung beim Packen. Listbox und Scrollbar geben

ihrer Packfunktion die zusätzliche Option -in und als Argument die Framevariable.

```
#!/usr/bin/perl -w
use strict;
use Tk;

my
    $mw = MainWindow->new;
my$

titel =
    $mw->Label(-text=>"Oben",                -relief=>'groove');
my$

framelist = liste =
    $mw->Listbox(-height => 4);
my$

scroll =
    $mw->Scrollbar(-command, [yview=>$
liste]);

    $liste->configure(-yscrollcommand => ['set',$
scroll]);
my $mw->Button(-text => "Links");
my $mw->Button(-text => "Rechts");

    $liste->insert(0,'eins','zwei','drei','vier','fünf');
# Anordnung der beteiligten Akteure$

titel->pack(-side,'top');

    $framelist->pack(-side,'top');$

liste->pack(-in=>
    $framelist, -side,'left');$

scroll->pack(-in=>
    $framelist, -side,'right');$

lbut->pack(-side,'left');

    $rbut->pack(-side,'right');
MainLoop;$
```

Nun sind die Elemente an den gewünschten Positionen. Beim näheren Betrachten stellen Sie aber fest, dass der Scrollbar sehr klein geraten ist. Und auch die Überschrift ist ein wenig zu kurz gekommen.



Ein Widget wird von sich aus nur soviel Platz in Anspruch nehmen, wie es benötigt. Diese Bescheidenheit bewirkt aber optische Unschönheiten, wenn beispielsweise drei unterschiedlich breite Buttons untereinander angeordnet sind. Hier kann man das Widget auffordern, den Raum in x-Richtung aufzufüllen, indem man als Argument für `-fill 'x'` angibt. Naheliegenderweise gibt es dann auch `'y'` für das senkrechte Ausfüllen. Das Argument `'none'` ist der Vorgabewert und füllt gar nichts aus.

Im Beispiel muss dem Scrollbar mitgeteilt werden, dass er bitte den vertikalen Raum füllen möge und den Titel müssen Sie anweisen, die gesamte Breite zu belegen. Die geänderten Zeilen lauten:

```
$titel->pack(-side,'top',-fill=>'x');  
$scroll->pack(-in=>  
    $framelist, -side,'right',-fill=>'y');$
```

Informationen

Für die Erstellung von grafischen Oberflächen mit Perl und Tk gibt es eine eigene Manpage. Ihr Aufruf lautet: `man perl/Tk`. Eine weitere Informationsquelle ist das Demoprogramm `wi dget`. Dies zeigt mehrere Beispiele für Widgets und die zugehörigen Quelltexte für Perl/Tk.

« [Kontrollelemente](#) | [Grafische Oberfläche: Tk](#) | [Informationsquellen](#) »

Informationsquellen

Mit dem Programmpaket Perl wird unter UNIX normalerweise eine umfangreiche Dokumentation in Form von Manpages installiert. Beim Aufruf von `man perl` bekommen Sie in erster Linie eine Übersicht über diese Manpages und was sie beinhalten.

Eine wichtige Informationsquelle im Internet ist die Webseite:

<http://www.perl.com>

Programmierwerkzeuge

Verschiedene Quellen tragen dazu bei, dass es unter UNIX eine Vielfalt an Werkzeugen für den Programmierer gibt. Mit UNIX zusammen wurde die Sprache C entworfen, um UNIX portabel zu machen. Alle Tools, die man für C brauchte, entwickelten sich also zunächst unter UNIX.

Im Universitätsumfeld entwickelten Studenten die Werkzeuge, die sie zum Arbeiten brauchten und fügten sie hinzu. In den letzten Jahren ist es vor allem die Open Source Bewegung, die neue Software für UNIX entwickelt.

-
- C-Compiler
 - make
 - Debugger
 - dbx
 - adb (System V)
 - gdb GNU debug
 - Analysewerkzeuge
 - Systemaufrufe verfolgen: strace und ltrace
 - Speicherlecks und -überläufe
 - Versionsverwaltung
 - SCCS (Source Code Control System)
 - RCS (Revision Control System)
 - Zusammenspiel mit make
 - CVS (Concurrent Versions System)
 - UNIX als CVS Server
 - Diverse Programmierhelfer
 - Kurzbetrachtung: lex und yacc
 - Unterschiede zwischen Textdateien: diff
 - Dateien aufs Byte geschaut
-

C-Compiler

Lange Zeit wurde jeder UNIX Maschine immer auch ein C-Compiler beigelegt. Selbst wenn dieser in den späteren Jahren als Handwerkszeug für den Softwareentwickler aufgrund mangelnder Aktualität wenig brauchbar war, benötigte ihn der Administrator, um einen neuen Kernel zu erstellen. Inzwischen ist es längst üblich, dass man den C-Compiler zu UNIX kaufen muss.

Dies, die Qualität manches Compilers und die Inkompatibilität zwischen den Compilern führte dazu, dass inzwischen der GNU-Compiler ein wichtiger Standard auf den UNIX-Maschinen ist. Er ist für fast jede Plattform verfügbar, er verhält sich überall in etwa gleich und ist immer auf dem neuesten Stand. Dazu kommt, dass er kostenlos ist. Das klingt fast nach einem Perpetuum Mobile der Informatik. Der Hintergrund sind die Universitäten, die mit diesem Compiler ein Grundgerüst zur Verfügung haben, um Studenten die Prinzipien des Compilerbaus zu lehren. Ergebnisse der Compilerbauforschung gelangen über Diplomarbeiten sehr schnell in die Weiterentwicklung.

Der UNIX-C-Compiler wird immer als `cc` aufgerufen. Dabei erkennt er selbst, ob er als Präcompiler, als Compiler oder als Linker tätig werden soll. Selbst ob es sich um C oder C++ handelt erkennt er an der Extension der Dateien. Obwohl hinter dem `cc` sehr unterschiedliche Compiler verschiedenster Hersteller stecken können, sind die wichtigsten Optionen überall gleich.

Wer vom PC oder Mac kommt, wundert sich vielleicht, dass sich unter UNIX keine IDE (Integrated Development Environment) durchgesetzt hat. Tatsächlich gibt es diese Umgebungen natürlich auch für X, wie beispielsweise das Apex unter Solaris. Dennoch arbeiten die meisten UNIX Programmierer von der Shell aus. Durch das sehr leistungsfähige `make` (siehe S. `make`) braucht das Übersetzen nur ein Kommando und man kann Abhängigkeiten definieren, die mit IDEs kaum machbar sind.

Für einen ersten Versuch soll ein einfaches Programm übersetzt und gestartet werden. Es heißt `moin.c` und grüßt die RegionDie weltmännischeren Programmierer können auch gern die Welt grüßen.

```
gaston> cat moin.c
main()
{
    puts("Moin, Torfmoorholm!");
}
gaston> cc moin.c
gaston> a.out
Moin, Torfmoorholm!
gaston>
```

Der C-Compiler wird mit `cc` gefolgt von dem Dateinamen der Quelldatei aufgerufen. Es entsteht die ausführbare Datei `a.out`. Mit Hilfe der Flag `-o` kann der Name der Zielfeldatei geändert werden. `cc` ruft nacheinander Precompiler, C-Compiler, Assembler und Linker auf. Wichtige Optionen beim Aufruf des C-Compilers lauten:

- `[-o Dateiname]`
Die Ausgabedatei erhält den Namen `Dateiname`.
- `[-c Dateiname]`

Kompiliert eine einzelne C-Quelltextdatei. Es entsteht eine linkfähige Objektdatei mit der Endung .o.

- [-IPfad]
Ergänzt den Pfad, in dem nach Headerdateien gesucht werden soll.
- [-LPfad]
Ergänzt den Pfad, in dem nach Bibliotheken gesucht werden soll.
- [-IName]
Verwendet beim Linklauf die Bibliothek `libName.a`. Die Datei wird in den Standardlinkpfaden (beispielsweise `/usr/lib`) und denen durch `-L` angegebenen gesucht.
- [-g]
Dem Code werden Informationen für den Debugger hinzugefügt. So können Sie im Debugger Variablen und Funktionen über ihre Namen ansprechen.
- [-DName]
Mit dieser Option können Namen definiert werden. Diese sind mit denen der Präprozessoranweisung `#define` gleichwertig. Soll der Name einen Wert zugewiesen bekommen, muss `-DName=Wert` angegeben werden.

Weil es so leicht zu verwechseln ist, sei es hier noch einmal auseinander gehalten. Die Option `-I` ist ein großes `i` (für `include`) und gibt den Headerpfad an und die Option `-l` ist ein kleines `L` (für `library`) und gibt die hinzuzubindende Bibliothek an.

Ein paar kleine Fallstricke gibt es, auf die Programmierer, die von anderen Umgebungen her kommen, unter UNIX leicht hereinfliegen können.

1. UNIX unterscheidet streng zwischen Groß- und Kleinschreibung. Die Datei `Test.H` ist nicht gleich `test.h`.
2. UNIX verwendet den Schrägstrich und nicht den Backslash als Pfadtrenner.
3. Unter UNIX ist es ein Unterschied, ob der Dateiname hinter `#include` in spitzen Klammern oder den Anführungszeichen gesetzt wird.

```
#include "test1.h"  
#include <test2.h>
```

`test1.h` wird zuerst im Sourceverzeichnis gesucht, ist also vom Programmierer geschrieben worden und gehört zum Projekt. `test2.h` wird ausschließlich im Include-Path gesucht, ist also ein System-Header.

4. Der Linker verhält sich im Gegensatz zu den meisten PC-Compilern so, dass er nur dann Funktionen aus Bibliotheken dazubindet, wenn deren Name als noch nicht aufgelöst offen sind. Damit hat die Reihenfolge der Dateien und Bibliotheken eine Bedeutung. Aus diesem Grund werden die Bibliotheken zum Schluß genannt.

« [Programmierwerkzeuge](#) | [Programmierwerkzeuge](#) | [make](#) »

make

Sobald ein Programm aus mehreren Dateien besteht, ist schnell der Punkt erreicht, wo es sinnvoll ist, sich einen kurzen Batch für das Compilieren zu schreiben. Statt dessen sollten Sie lieber ein Makefile schreiben. Das ist nicht viel aufwändiger, hat aber den Vorteil, dass das Programm make überwacht, welche Änderungen bei der neuesten Version zu berücksichtigen sind. Das heißt, dass make das Übersetzen nicht öfter veranlasst, als erforderlich ist.

Das Projekt meinprog besteht aus den Sourcedateien `haupt.c`, `test.c` und `tools.c`. Jede dieser Dateien hat eine Headerdatei (`haupt.h`, `test.h` und `tools.h`), die es jeweils selbst einbindet. Dazu bindet `haupt.c` jede andere Headerdatei ein und jedes Modul bindet die globalen Definitionen aus `haupt.h` ein.

Eine Datei Makefile wird angelegt und darin wird der Weg der Compilierung beschrieben. Das Programm meinprog hängt von den Dateien `test.o`, `haupt.o` und `tools.o` ab. Das formuliert man in einem Makefile folgendermaßen:

```
meinprog: test.o haupt.o tools.o
```

Mit dieser Zeile wird die Abhängigkeit der Datei `meinprog` von den Dateien `test.o`, `haupt.o` und `tools.o` definiert. Man bezeichnet `meinprog` als Ziel oder englisch `target`. Die darauf folgenden Zeilen beschreiben, wie die Zieldatei hergestellt wird. Die Datei `meinprog` wird generiert, indem der Compiler mit den Objektdaten als Parameter aufgerufen wird. Mit Hilfe der Option `-o` legen Sie fest, dass das Ergebnis `meinprog` heißen soll. Der Compiler merkt von sich aus, dass er hier nur linken soll, da keine echten Quellcodedateien vorhanden sind. Solche Aktionszeilen müssen mit einem Tabulator beginnen. Es dürfen keine Leerzeichen verwendet werden.

```
meinprog: test.o haupt.o tools.o
    cc -o meinprog test.o haupt.o tools.o
```

Wer nun einfach `make` aufruft, erhält überraschenderweise bereits ein komplett übersetztes Programm. Auf dem Bildschirm erscheinen folgende Zeilen:

```
gaston> make
cc      -c -o test.o test.c
cc      -c -o haupt.o haupt.c
cc      -c -o tools.o tools.c
cc -o meinprog test.o haupt.o tools.o
gaston>
```

Tatsächlich »weiß« `make`, wie man aus `c`-Dateien `o`-Dateien macht und da er mit den Regeln des Makefiles die `o`-Dateien nicht erzeugen kann, sieht `make` im aktuellen Verzeichnis nach, ob es Dateien gibt, aus denen er `o`-Dateien

generieren kann.

Ändern Sie die Datei test.c, so wird nur test.c übersetzt und meinprog neu gebunden. Ändern Sie allerdings test.h, passiert nichts. make kennt die Abhängigkeiten der Header nicht. Damit make auch Änderungen der Headerdateien überwacht, ist folgende Änderung im Makefile notwendig.

```
test.o : test.c test.h haupt.h
```

```
tools.o : tools.c tools.h haupt.h
```

```
haupt.o : haupt.c haupt.h test.h tools.h
```

Eine vollständige Aktionszeile müsste hier eigentlich etwa so lauten:

```
test.o : test.c test.h haupt.h
    cc -c -o test.o test.c
```

Da aber make die erforderliche Aktion bereits kennt, braucht man sie nicht zu erwähnen.

Das Programm make ist also ein Tool, um Zielfdateien aus den Quelldateien mit minimalem Aufwand zu generieren. Dazu wird in einer Datei namens makefile oder Makefile, die Abhängigkeiten der Dateien beschrieben und die Programmaufrufe festgelegt, die aus den jeweiligen Quellen die Ziele generieren. make erkennt, wenn eine der Quelldateien neuer als die Zielfdatei ist, und ruft die Generierungsprogramme auf, bis die Zielfdateien neuer als die jeweiligen Quellen sind oder eine Aktion scheitert.

Ein Makefile hat Einträge der folgenden Grundstruktur:

**Ziel : Abhängigkeiten
Generierungskommando**

Diese Grundstruktur nennt man Regel. Eine neue Regel muss mit einer Leerzeile von der vorherigen getrennt werden. Der Leerraum vor dem Generierungskommando muss ein Tabulatorzeichen sein. Es können auch mehrere Kommandozeilen nacheinander angegeben werden. Alle müssen mit Tabulator eingerückt sein. Die Kommandozeilen werden jeweils in einer separaten Shell abgearbeitet. In manchen Situationen gibt das Seiteneffekte, die Sie berücksichtigen müssen. Beispiel:

```
try :
    cd .. ; pwd
    pwd
```

Die Ergebnisse der beiden Aufrufe von `pwd` sind nicht gleich. Der Wechsel mit `cd ..` gilt nur für die aktuelle Zeile. In der nächsten Zeile wird wieder aus dem bisherigen Verzeichnis gearbeitet.

```
cd .. ; pwd
/home/arnold/my/src/unix
pwd
/home/arnold/my/src/unix/make
```

Hängen also Kommandos so zusammen, dass sie in einer gemeinsamen Shell bearbeitet werden müssen, sollten sie in dieselbe Zeile geschrieben werden und mit Semikola getrennt werden. Bei langen Zeilen kann mit einem Backslash die Zeile in der nächsten Zeile fortgesetzt werden.

Als Kommentarzeichen gilt das `#` in der ersten Spalte.

Makros im Makefile

Durch die Verwendung von Makros können die Makefiles besser strukturiert und flexibler gestaltet werden. Makros sind Variablen, denen Zeichenketten zugewiesen werden und dann durch Voranstellen eines `$`-Zeichens ausgewertet werden. Durch das Zusammenfassen von Dateinamen oder Optionen kann ein Makefile kürzer und vor allem besser lesbar werden.

Im Beispiel werden die Objektdaten zusammen behandelt und zweimal aufgezählt, einmal in der Abhängigkeitsbeschreibung von `meinprog` und dann im Compileaufruf.

```
meinprog: test.o haupt.o tools.o
        cc -o meinprog test.o haupt.o tools.o
```

Hier können Sie eine Variable `OBJS` definieren, die die Objekte bezeichnet. Durch Einsetzen von `OBJS` ergibt sich die folgende Makedatei.

```
OBJS = test.o haupt.o tools.o
```

```
meinprog: $(OBJS)
        cc -o meinprog
```

Die Variablen müssen nicht im Makefile selbst definiert werden. `make` kann auf Umgebungsvariablen zurückgreifen, die von der aufrufenden Shell festgelegt wurden.

Vordefinierte Makros

Es ist möglich, mehrere Ziele mit einer Regel zu behandeln. So könnte beispielsweise `$(OBJS)` als Ziel verwendet werden. Die einzelnen Ziele werden nacheinander aufgelöst. Im Generierungskommando kann auf das aktuelle Ziel Bezug genommen werden. Dazu gibt es vordefinierte Makros, die hier am Beispiel `haupt.o` aufgezeigt werden.

[Vordefinierte make-Makros]L|L Makro & Bedeutung

`$@` & Dateiname des Ziels (haupt.o)

`$*` & Basisnamen des Ziels (haupt)

Suffixregeln

Die Suffixregeln beschreiben den Übergang einer Dateiendung zu einer anderen. Eine solche Regel erkennen Sie daran, dass das Ziel die zwei Dateiendungen mit dem jeweiligen Punkt am Anfang direkt hintereinander stehen hat.

```
. Quell.ziel:
```

Der typischste Übergang ist sicher der von C-Quellen zu Objekten. Die Quellen enden auf `c` und die Objektdateien auf `o`. Die entsprechende Suffixregel lautet dann:

```
.c.o:
    cc -c $<
```

Das interne Makro `$<` darf nur bei Suffixregeln verwendet werden und bezeichnet das aktuelle Ziel.

Mehrere Ziele

Ein Makefile kann mehrere Programme generieren. Das wird eingesetzt, wenn gleiche Quelltexte für mehrere Projekte gebraucht werden, die vielleicht sogar noch voneinander abhängig sind. Ein typisches Beispiel sind Client- und Serverprogramme, die in den Headern gleiche Datenstrukturen verwenden.

```
all: client server
```

```
client: (COMMONOBS)
    $(CLTOBS)
    ...
\par server:$(
    (SENDHEADER) (SRVOBS)
    ...
```

Das erste Ziel ist immer das Ziel des gesamten Makefiles. In diesem Fall würde also beim Aufruf von `make` erst das Ziel `all` generiert werden. Da es keinerlei verbundene Aktion gibt, wird lediglich geprüft, ob die Abhängigkeiten erfüllt sind. Entsprechend wird als nächstes das Ziel `client` und dann das Ziel `server` gebildet. Es ist nicht zwingend, aber üblich, das Pseudoziel, das alle Programme eines Makefile generiert, `all` zu nennen.

Oft werden Makefiles auch zur Installation verwendet. Dazu wird ein Pseudoziel `install` eingeführt, das überprüft, ob alle Dateien des Projektes an den richtigen Stellen vorhanden sind und ansonsten als Aktion einfache Kopierbefehle absetzen. Aufgerufen wird so eine Installation mit `make` und als Parameter das Ziel `install`.

```
make install
```

Unterabschnitte

- [Makros im Makefile](#)
- [Vordefinierte Makros](#)
- [Suffixregeln](#)
- [Mehrere Ziele](#)

« [C-Compiler](#) | **[Programmierwerkzeuge](#)** | [Debugger](#) »

Debugger

Ein Debugger ist ein Programm, das den Programmierer dabei unterstützen soll, Programmfehler zu finden. Diese Fehler werden als Bug (engl. für Wanze oder Käfer) bezeichnet. Entsprechend ist der Debugger ein »Entwanzer«. Ein Debugger lässt das zu untersuchende Programm unter seiner Kontrolle ablaufen. Sie können das Programm an bestimmten Punkten stoppen lassen und im Einzelschritt verfolgen. Dabei lassen sich Variableninhalte anzeigen. Damit das erzeugte Programm dem Debugger etwas zu sagen hat, muss es mit dem Flag `-g` kompiliert worden sein.

Äußert sich das Problem in einem Programmabsturz, so wird man typischerweise das Programm im Debugger starten und in die Situation des Zusammenbruchs laufen lassen. Sobald das Programm zusammenbricht, fängt es der Debugger auf und meldet, an welcher Stelle der frühe Tod zu vermeiden war.

Schwieriger ist es schon, wenn das Programm durchaus läuft, aber die Ergebnisse nicht die erwarteten sind. In solchen Fällen setzen Sie vor dem Start des Programms an den Stellen, wo Sie Probleme vermuten jeweils einen Breakpoint. Nach dem Start des Programmes wird dieses normal laufen, bis es einen Breakpoint erreicht. Dort wird das Programm anhalten. Sie können sich die Variablen anzeigen lassen und Schritt für Schritt weitergehen.

Moderne Debugger sind sogar in der Lage, einen bereits laufenden Prozess unter ihre Kontrolle zu bekommen. Das ist besonders deswegen interessant, weil es ansonsten gar keine Möglichkeit gäbe, beispielsweise Dämonen zu debuggen.

Der Debugger ist bei der Analyse eines core dump hilfreich. Ein core dump entsteht dann, wenn das Programm vom Betriebssystem die Arbeitserlaubnis entzogen bekommen hat, weil es beispielsweise versucht hat, in Speicherbereiche zu greifen, die ihm nicht gehören. Dabei wird der Abzug des Speichers des Prozesses in jenem Moment in eine Datei namens core geschrieben.. Ein Debugger kann mit Hilfe dieser Datei feststellen, an welcher Zeile der Absturz erfolgt ist und welche Funktionen das Programm durchlaufen hat, um an diese Stelle zu kommen. Da der Debugger auch in der Lage ist, die Variableninhalte anzuzeigen, können Sie mit dieser Information recht gut feststellen, was die Ursache war.

Für die Debugger gibt es teilweise auch grafische Oberflächen. Vom Prinzip her arbeiten diese natürlich gleich, sind aber teilweise etwas leichtfüßiger zu bedienen.

Unterabschnitte

- [dbx](#)
- [adb \(System V\)](#)
- [gdb GNU debug](#)

dbx

Der Debugger dbx läuft auf diversen Plattformen wie Sun oder AIX. Beim Aufruf des Debuggers kann als Parameter ein ausführbares Programm angegeben werden. Als dritter Parameter kann noch ein core angegeben werden.

dbx Programm **dbx Programm core**

Programm ist das zu untersuchende Programm. core heißt die Datei, die UNIX bei einem Programmzusammenbruch erzeugt. Soll sie analysiert werden, muss sie als zweiter Parameter angegeben werden. Innerhalb des Debuggers können die folgenden Befehle gegeben werden:

- [where] zeigt den Stack der aufgerufenen Funktionen an. Das funktioniert auch, wenn man einen core dump geladen hat. Damit wissen Sie, wo das Programm zusammengebrochen ist.
- [stop] setzt einen Breakpoints. Dabei gibt es zwei Formen des Befehls. stop at erwartet als Parameter eine Zeile und stop in eine Funktion.
- [delete] entfernt einen Breakpoint.
- [run] startet das geladene Programm.
- [cont] setzt den Lauf fort, wenn das Programm durch einen Breakpoint gestoppt worden war.
- [step] durchläuft schrittweise das Programm und zeigt immer die aktuelle Sourcecodezeile
- [display variable] beobachtet Variablen. Nach jedem step wird die Variable mit ihrem neuen Wert angezeigt
- [quit] verlässt den Debugger.

adb (System V)

adb ist ein älterer Debugger, der bei System V mitgeliefert wird. Da aber Softwarewerkzeuge nicht so oft wie Netzwerksoftware aktualisiert wird, kann es Ihnen durchaus passieren, dass er Ihnen einmal begegnet. Der Debugger verwendet Buchstaben als Kommandos.

Auch mit dem adb läßt sich ein coredump analysieren. Der Aufruf erfolgt wie bei anderen Debuggern mit dem Programm und dem core dump als Parameter. Anschließend wird mit c die Liste der Aufrufe erzeugt und mit q kann der Debugger verlassen werden.

Einen Breakpoint setzen Sie, indem Sie zunächst die Adresse nennen, an der der Breakpoint sein soll. Dann muss ein ein Doppelpunkt und das Kommando b folgen. Die Adresse kann ein Funktionsname mit einem Offset oder eine hexadezimale Position sein. Beispiel:

```
tausche+0x26:b
```

Der Befehl b allein zeigt die Liste der definierten Haltepunkte an. Mit dem Befehl r wird das geladene Programm gestartet. Das Programm läuft dann bis zum nächsten Haltepunkt. Dort kann man mit c weiterlaufen, mit s schrittweise weitergehen oder mit k stoppen.

[Befehle des adb]C|L Kommando & Wirkung

b & zeige die Breakpoints

r & starte das Programm

c & weiterlaufen (continue), bei core dump: zeige die Aufruffolge

s & schrittweise ablaufen lassen

k & stoppe das Programm

« [dbx](#) | [Debugger](#) | [gdb GNU debug](#) »

gdb GNU debug

Der GNU Debugger gdb wird normalerweise mit dem zu untersuchenden Programm als Parameter gestartet. Als weiterer Parameter kann auch hier ein core dump angegeben werden.

gdb kann ein laufendes Programm unter seine Fittiche nehmen. Dazu wird der gdb gestartet und der Befehl `at` oder `attach` abgesetzt. Der Prozess wird gestoppt und mit dem gdb verbunden.

```
attach PID
```

Alternativ können Sie die PID auch als zweiten Parameter beim Aufruf von gdb angeben. Hier folgt ein Beispiel:

```
gaston> ps -ef | grep one
arnold   1339  1223  0 11:37 pts/3    00:00:00 one
gaston> gdb one 1339
GNU gdb 20010316
Copyright 2001 Free Software Foundation, Inc.
/home/arnold/my/src/unix/ipc/1339: No such file or directory.
Attaching to program: /home/arnold/src/unix/one, process 1339
Reading symbols from /lib/libc.so.6...done.
Loaded symbols for /lib/libc.so.6
0x400f3d64 in read () from /lib/libc.so.6
(gdb) next
Single stepping until exit from function read,
...
```

Dieses etwas gekürzte Protokoll zeigt, wie der Prozess `one` unter die Kontrolle des Debuggers geholt wird. Zunächst wurde die PID von `one` ermittelt und daraufhin gdb gestartet. Sie können sehen, wie die Meldung des Copyright erscheint. Anschließend versucht gdb, die Datei 1339 zu öffnen. Als das nicht funktioniert, versucht er, den entsprechenden Prozess zu bekommen, was ihm auch gelingt.

Im Debugger gibt es diverse Befehle. Mit `run` können Sie das geladene Programm starten, mit dem Befehl `kill` wieder stoppen. Mit `quit` verlassen Sie den Debugger.

Um an den kritischen Stellen den Programmablauf unterbrechen zu können, setzen Sie einen Breakpoint. Der Befehl `break` erwartet als Parameter die Zeilenzahl im Listing oder einen Funktionsnamen. Die Zeilennummer ermitteln Sie durch Auflisten des Programms, was gdb durch den Befehl `list` ausführt.

Haben Sie den Breakpoint erreicht, können Sie mit den Befehlen `step` oder `next` Schritt für Schritt das Programm durchlaufen. Dabei wird `step` in aufgerufene Funktionen hinuntersteigen, was `next` nicht tut. Mit `continue` lassen Sie das Programm weiterlaufen.

Wollen Sie einen Blick auf Variablen werfen, ist `print` dazu der einfachste Befehl. Mit dem Befehl `watch` können Sie Variablen unter Bewachung stellen. Dieser Befehl zeigt den Inhalt, sobald er sich ändert.

[Kommandos des GNU Debuggers]L|L Kommando & Aktion

quit & Debugger verlassen

run & das geladene Programm starten

kill & stoppen des laufenden Programms

print Variable & Inhalt der Variablen anzeigen

watch Variable & Variable beobachten

list & zeigt einen Ausschnitt aus dem Sourcelisting

break Funktionsname & setzen eines Breakpoints

break Zeilennummer & setzen eines Breakpoints

clear Funktionsname & löscht einen Breakpoint

clear Zeilennummer & löscht einen Breakpoint

next & eine Zeile weitergehen

step & eine Zeile weiter und ggf. in eine Funktion hinein

continue & fortsetzen des angehaltenen Prozesses

« [adb \(System V\)](#) | [Debugger](#) | [Analysewerkzeuge](#) »

Analysewerkzeuge

Neben den Debuggern, die natürlich auch der Analyse dienen, gibt es eine Reihe Werkzeuge, die den Ablauf von Programmen verfolgen. Teils dienen diese Werkzeuge dazu, die Performance eines Programmes zu ermitteln und zu verbessern. Teils suchen sie nach schleichenden Fehlern wie Speicherlecks. Hier werden einige dieser Werkzeuge herausgegriffen ohne Anspruch auf Vollständigkeit.

Unterabschnitte

- [Systemaufrufe verfolgen: strace und ltrace](#)
 - [Speicherlecks und -überläufe](#)
 - [Die Bibliothek Electric Fence](#)
 - [Dmalloc und LeakTracer](#)
-

Systemaufrufe verfolgen: strace und ltrace

Das Tool `strace` wird mit einem Programm als Parameter gestartet und zeigt dessen Systemaufrufe inklusive der Parameter und Rückgabewerte an.

Mit `ltrace` lassen sich Bibliotheksaufrufe verfolgen. Beide Programme erfordern nicht, dass das zu beobachtende Programm vorher mit Debug-Option compiliert wurde.

Unterabschnitte

- [Die Bibliothek Electric Fence](#)
- [Dmalloc und LeakTracer](#)

Speicherlecks und -überläufe

Fehler in der Speicherverwaltung sind schwer zu finden. Das Phänomen eines »Speicherlecks« bedeutet, dass Speicher angefordert wird, aber nicht mehr freigegeben wird. So ein Fehler ist bei manchen einfachen Anwendungsprogrammen fast kein Problem. Kaum jemand würde es je bemerken. Läuft aber das Programm als Dämon auf einer Maschine, die vielleicht monatelang ohne Unterbrechung läuft, entzieht das Programm auf lange Sicht allen anderen Programmen den Speicher und die Maschine wird immer langsamer.

Ein anderes Problem ist der Zugriff über die Speicherränder. Wurde eine Zeichenkette von 120 Byte Länge in einen Speicher kopiert, der nur eine Länge von 100 Byte hat, befinden sich 20 Byte in Speicherbereichen, wo sie nichts verloren haben und ggf. andere Variablen überschreiben. Man spricht hier vom Buffer Overrun. Da man nie weiß, welche Variablen verändert wurden und an welcher Stelle im Programm diese verwendet werden, kommt ein Kollaps fast immer überraschend. Auch die Wirkung ist schwer vorhersehbar, da sich nicht abschätzen lässt, welche Daten zerstört werden.

Die Bibliothek Electric Fence

Electric Fence (übersetzt etwa Elektrozaun) überprüft, ob die Grenzen von Puffern überschritten werden und ob Speicherbereiche verwendet werden, die in der Zwischenzeit wieder frei gegeben wurden.

Electric Fence ist eine Bibliothek, die beim Generieren des Programmes hinzu gebunden wird und sich an die Stelle der Funktionen setzt, die normalerweise die Speicherverwaltung durchführen. Der einzige Unterschied zur normalen Entwicklung ist also das Hinzubinden der Bibliothek.

```
cc -o fehler fehler.c -lefence
```

Das Programm wird danach aus einem Debugger gestartet. Bei erkannten Verletzungen wird ein Signal ausgelöst, das einen Zusammenbruch auslöst. Der Debugger kann dann leicht mit einem entsprechenden Kommando (where beim gdb) feststellen, wo dieser Fehler aufgetreten ist.

Quelle: <http://perens.com/FreeSoftware>

Dmalloc und LeakTracer

Auch für die Erkennung von Speicherlecks gibt es Programmierhilfen. Man findet sie im Internet, indem man in den Suchmaschinen nach Begriffen wie »memory leak« sucht. Die Werkzeuge arbeiten normalerweise als Bibliotheken, die die normalen Schnittstellen für das Anfordern und Freigeben von

Speicher anbieten. Dabei protokollieren sie, ob alle angeforderten Speicherbereiche auch korrekt wieder freigegeben werden.

Hier sind zwei Pakete exemplarisch genannt. `dmalloc` ist ein Tool für C-Programme, die ihren Speicher mit Hilfe der Funktion `malloc()` anfordern und mit der Funktion `free()` freigeben, bzw. freigeben sollten. Der `LeakTracer` ist für die C++-Programmierer interessanter, da er die Aufrufe `new()` und `delete()` kontrolliert.

Beide Programme sind kostenlos im Web zu bekommen. Quelle:

<http://dmalloc.com>

<http://www.andreassen.org/LeakTracer/>

« [Systemaufrufe verfolgen: strace und](#) | [Analysewerkzeuge](#) | [Versionsverwaltung](#) »

Versionsverwaltung

Versionsverwaltungen sind unentbehrliche Tools in der professionellen Softwareerstellung. Solche Systeme sorgen dafür, dass alte Entwicklungsstände archiviert werden, und dass es keine Reibungsverluste gibt, wenn mehrere Entwickler an einer Software gemeinsam Änderungen durchführen.

Unterabschnitte

- [SCCS \(Source Code Control System\)](#)
 - [RCS \(Revision Control System\)](#)
 - [Zusammenspiel mit make](#)
 - [CVS \(Concurrent Versions System\)](#)
 - [Umgebungsvariable CVSROOT](#)
 - [Anlegen eines Repositories](#)
 - [Dateien aus dem Repository holen](#)
 - [Änderungen im Repository abstellen](#)
 - [Neue Dateien hinzufügen](#)
 - [Dateien aus dem Repository entfernen](#)
 - [Rückgriff auf alte Versionen](#)
 - [Releases](#)
 - [UNIX als CVS Server](#)
 - [UNIX als Client](#)
 - [Windows-Client WinCVS](#)
-

SCCS (Source Code Control System)

SCCS ist ein kommerzielles System, das einigen Compilern beiliegt. Die Sourcen liegen auf einem zentralen Dateisystem, der für alle beteiligten Programmierer zugänglich ist. Für jede Datei, die SCCS kontrolliert, wird eine SCCS-Datei angelegt. Ihr Name bildet sich aus dem der zu verwaltenden Datei und hat den Präfix »s.«. Darin werden die Differenzen der verschiedenen Stände gespeichert.vgl. Detering, Reinhard: UNIX Handbuch System V. Sybex, Düsseldorf-San Franzisko-Paris-London-Soest, 4. Aufl., 1990. S. 637ff. und Illik, J. Anton: Programmieren in C unter UNIX. Sybex, Düsseldorf, 1992. S. 652ff.

Eine Datei wird mit dem Befehl `admin` unter SCCS gestellt. Sie erhält dabei die Standardversionsnummer 1.1 und ist nach Ausführung des Befehls erst einmal verschwunden. Statt ihrer gibt es nun die SCCS-Datei, die den Präfix »s.« vor dem Namen trägt.

Wollen Sie die Datei nur lesen, erhalten Sie mit dem Befehl `get` eine schreibgeschützte Version aus der SCCS-Datei.

Sobald Sie eine Änderung machen wollen, checken Sie mit dem Kommando `get -e` die entsprechende Datei aus. Dabei werden Sie automatisch zum Besitzer dieser Datei und erhalten Schreibrecht. Eine Datei mit Schreibrecht ist aus Sicht des SCCS immer ausgecheckt und bildet eine Sperre gegen erneutes Auschecken. Damit ist gewährleistet, dass nur ein Programmierer gleichzeitig die Datei verändern kann.

Bei jedem Auschecken wird die Nummer hinter dem Punkt der Releasenummer inkrementiert. Die nächste Version wäre also 1.2. Wollen Sie einen Release wechseln, also die Nummer vor dem Punkt erhöhen, geben Sie beim `get -e` auch den Parameter `-r` gefolgt von der neuen Releasenummer an.

Sind die Änderungen durchgeführt, wird die Datei mit dem Befehl `delta` wieder eingchecked. Dabei werden alle Änderungen in die SCCS-Datei übernommen. Die Originaldatei verschwindet wieder. Vorher erscheint ein Prompt und bittet um einen Kommentar, in dem Sie festhalten sollten, welche Änderung Sie gemacht haben.

Sollen die Änderungen doch nicht in das System gelangen, können Sie mit dem Befehl `unget` die Datei wieder in den Zustand zurückversetzen, der vor dem Auschecken war.

[Übersicht über die SCCS-Kommandos]
L|L Befehl & Wirkung
delta Datei & Datei einchecken
get Datei & Datei zum Lesen aus dem SCCS holen
get -e Datei & Datei zum Verändern auschecken
unget Datei & Auschecken revidieren

RCS (Revision Control System)

RCS gehört zur GNU Software und ist damit frei erhältlich. Es ähnelt in vieler Hinsicht SCCS. Auch hier werden die Änderungen in separaten Dateien abgestellt. Allerdings werden die RCS-Dateien in einem eigenen Verzeichnis namens RCS abgelegt.vgl. Husain/Parker et al: Red Hat Linux Unleashed. SAMS, Indianapolis, 1996. pp. 890

Nachdem im Sourceverzeichnis das Verzeichnis RCS angelegt wurde, nimmt der Befehl `c i` (check in) eine Datei unter RCS-Kontrolle. Mit dem gleichen Befehl werden später geänderte Dateien wieder in das System eingchecked. Das System bittet dabei um eine Kommentierung der Änderung. Wie beim SCCS verschwindet dann die Originaldatei und alle Informationen liegen in der RCS-Datei, die im Verzeichnis RCS liegt und wie die Originaldatei heißt, allerdings »,v« angehängt bekommt.

Der Befehl `co` holt die letzte Version zum Lesen aus dem RCS heraus. Erst `co -l` erzeugt eine Version, die änderbar ist. Ein paralleles Bearbeiten der Datei ist dann nicht mehr möglich, bis die Datei mit `c i` wieder unter die Verwaltung von RCS gestellt wird.

Neue Releases werden durch den Parameter `-r` beim Einchecken angelegt. Soll der Unterschied zwischen zwei Versionen ermittelt werden, wird dazu der Befehl `rcsd i ff` verwendet. Ohne Parameter werden die ausgecheckte Version mit der zuletzt eingcheckedten Version verglichen. Es können aber auch beliebige Versionen verglichen werden. Die Versionen werden durch `-r` angegeben.

Sie können jeder Datei zuordnen, welcher Programmierer sie ändern darf. Dazu wird der Verwaltungsbefehl `r cs` mit der Option `-a` gefolgt von den Namen der Programmierer aufgerufen. Mit der Option `-e` kann einzelnen Programmierern das Recht entzogen werden, die Datei zu ändern.

Zusammenspiel mit make

Durch Ergänzungen im Makefile können Sie erreichen, dass zum Compilieren die aktuellen Versionen aus dem RCS verwendet werden. Die Suffixregel wird so gesetzt, dass die Objektdaten nicht aus dem C Quellcode generiert wird, sondern aus der Extension der RCS-Datei. Dafür muss in der ersten Aktionszeile die Datei zum Lesen ausgecheckt werden.

```
.C,v.o:
    co
$<
    cc -c$
*.C
    rm -f
```

Die Tilde (~) verhindert, dass make abbricht, wenn rm misslingt.

Unterabschnitte

- [Umgebungsvariable CVSROOT](#)
 - [Anlegen eines Repositories](#)
 - [Dateien aus dem Repository holen](#)
 - [Änderungen im Repository abstellen](#)
 - [Neue Dateien hinzufügen](#)
 - [Dateien aus dem Repository entfernen](#)
 - [Rückgriff auf alte Versionen](#)
 - [Releases](#)
-

CVS (Concurrent Versions System)

CVS (Concurrent Versions System) dient dazu, bei der Softwareentwicklung die Fortentwicklung der Projekte zu speichern. Es ermöglicht einerseits den Zugriff auf ältere Versionen und andererseits den unproblematischen Zugriff durch mehrere Programmierer.

Während RCS (Revision Control System) und SCCS nur einzelne Dateien kontrollieren, arbeitet CVS projektorientiert. Es wird pro Projekt ein Repository angelegt, das mehrere Module (Verzeichnisse) und darunter wieder mehrere Dateien aufnehmen kann. Auch wenn es möglich ist, mehrere Projekte in einem einzigen Repository als Module abzulegen, wird dies hier nicht näher betrachtet. Die minimale Ersparnis steht in keinem Verhältnis zum entstehenden Durcheinander und dem Verlust an Flexibilität.

CVS kann auch als echte Client-Server-Applikation installiert werden. Das hat mehrere Vorteile. Damit ist die Versionskontrolle unabhängig von den Fähigkeiten des Netzwerkdateisystems. Darüber hinaus eignet sich CVS auf diese Weise sogar zum Betrieb über das Internet.

Umgebungsvariable CVSROOT

Das Repository ist der Ort, wo CVS die Sourcen und ihre Entwicklung zentral abspeichert. Hier hat kein Entwickler etwas zu suchen. Um die Sourcen weiter zu bearbeiten, legt sich jeder Entwickler an einer beliebigen anderen Stelle ein Arbeitsverzeichnis an und arbeitet dort.

Der Ort des Repository wird in der Environment-Variablen CVSROOT festgehalten. Unter UNIX kann diese in der /etc/profile oder in den jeweiligen lokalen rc-Dateien, beispielsweise .bashrc definiert werden.

```
export CVSROOT=/home/cvs
```

Anlegen eines Repositories

Das Verzeichnis, auf das CVSROOT zeigt, muss angelegt werden. Anschließend wird mit dem Befehl `cvs init` das Repository angelegt. Damit sind die internen Verwaltungsdateien erzeugt. Noch gibt es

aber keinen Inhalt. Ein Modul muss aber in jedem Fall eingetragen werden, da ansonsten keine Arbeitsumgebung aus dem Repository generiert werden kann.

Das Eintragen eines Moduls wird durch den Befehl `cvs import` erreicht. Bei kleineren Projekten ist es naheliegend, ein leeres Basisverzeichnis anzulegen. Besser ist eine thematische Untergliederung in Module bzw. Verzeichnisse.

```
export CVSR00T=/home/cvs
mkdir $CVSR00T
cvs init
mkdir worksrc
cd worksrc
cvs import -m "Basisverzeichnis" . v1_0 R1_0
```

Jetzt existiert ein Repository, aus dem sich jeder Programmierer an anderer Stelle ein Arbeitsverzeichnis erstellen kann. Das eben angelegte Verzeichnis `worksrc` wird nun nicht mehr benötigt und kann entfernt werden. Sie können es natürlich auch als Arbeitsverzeichnis weiterverwenden.

Dateien aus dem Repository holen

Um erstmalig mit den Sourcen arbeiten zu können, müssen Sie mit dem Befehl `cvs checkout` mindestens ein Arbeitsverzeichnis aus dem Repository erzeugen. Da oben bereits ein leeres Basisverzeichnis angelegt wurde, können Sie an beliebiger Stelle ein Arbeitsverzeichnis anlegen.

```
cvs checkout .
cvs checkout unix
```

In der ersten Zeile wird das Hauptverzeichnis eines Projektes mit allen Unterverzeichnissen in das Arbeitsverzeichnis übernommen. Die zweite Zeile holt nur das Modul `unix` und seine Unterverzeichnisse aus dem Projekt heraus.

Änderungen im Repository abstellen

Während bei anderen Versionssystemen die zu bearbeitenden Dateien reserviert werden müssen, ist dies bei CVS nicht erforderlich. Die geänderten Dateien werden mit dem Befehl `cvs commit` in das Repository übernommen. Sicherer als das Abstellen einzelner Dateien ist die Verwendung der Modulnamen, da CVS selbst merkt, was sich geändert hat.

Neue Dateien hinzufügen

Neue Dateien werden mit `cvs add` gefolgt vom Dateinamen im Repository angemeldet. Beim nächsten Einstellen der Änderungen wird diese Datei auch hinzugefügt.

Dateien aus dem Repository entfernen

Wenn eine Datei entfernt werden soll, muss sie zunächst im Arbeitsverzeichnis gelöscht werden. Dann kann sie mit dem Befehl `cvs remove` mit dem Dateinamen als Parameter aus dem Repository entfernt werden.

```
rm oldfile.cpp
cvs remove oldfile.cpp
cvs commit oldfile.cpp
```

Rückgriff auf alte Versionen

Mit dem Befehl `cvs diff` mit dem Dateinamen als Parameter können Sie sehen, was sich seit der letzten Version geändert hat. Dieser Befehl vergleicht die aktuell im Arbeitsverzeichnis vorliegende Version mit derjenigen, die im Repository liegt. Haben Sie die aktuelle Version schon abgestellt, können Sie angeben, mit welcher Version Sie die aktuelle vergleichen wollen. Dazu dient der Parameter `-r`. Schließlich ist es auch möglich, zwei alte Versionen miteinander zu vergleichen, indem Sie den Parameter `-r` zweimal angeben.

```
cvs diff myfile.cpp
cvs diff -r 1.1 myfile.cpp
cvs diff -r 1.1 -r 1.2 myfile.cpp
```

Welche Version einer Datei vorliegen, können Sie sich mit `cvs status` mit dem Dateiname als Parameter anzeigen lassen.

Releases

Soll eine Release der Software erstellt werden, besteht diese aus den verschiedensten internen Versionen des CVS. Um einen zusammengehörigen fertigen Stand zu markieren, gibt es den Befehl `cvs tag`, gefolgt von einem Bezeichner. Damit wird allen aktuell im Arbeitsverzeichnis abgestellten Dateien ein Vermerk angeheftet, anhand dessen Sie später jederzeit wieder diese Version herstellen können. Dazu wird ein `cvs checkout` Befehl mit der Option `-r` verwendet.

```
cvs tag alfa0_8
cvs checkout -r alfa0_8 .
```

« [Zusammenspiel mit make](#) | [Versionsverwaltung](#) | [UNIX als CVS Server](#) »

Unterabschnitte

- [UNIX als Client](#)
- [Windows-Client WinCVS](#)

UNIX als CVS Server

Anstatt die Verwaltung auf einem Netzlaufwerk durchzuführen, kann CVS auch als Client-Server Applikation installiert werden. Auf dieser Basis arbeiten viele Open-Source-Projekte sogar weltweit über das Internet zusammen.

Erster Schritt ist die Installation des CVS-Servers. Es wird ein zentrales Repository angelegt. Das dafür benötigte Verzeichnis kann überall liegen. Als Beispiel sei hier /home/cvs gewählt. In diesem Verzeichnis wird ein Verzeichnis CVSROOT (in Großbuchstaben) angelegt.

```
cvs -d /home/cvs init
```

Nun geht es daran, die Rechte korrekt zu setzen. Um neue Repositories einzurichten, muss jeder Entwickler das Recht haben, in /home/cvs ein Verzeichnis anzulegen. Dies erreichen Sie beispielsweise, indem Sie die folgenden Rechte zuordnen.

```
chgrp prog /home/cvs  
chmod 775 /home/cvs
```

Nun ist jedes Mitglied der Gruppe prog berechtigt, in diesem Verzeichnis neue Verzeichnisse anzulegen. Wer in der Gruppe prog ist, lässt sich leicht in der /etc/group steuern.

Der Zugriff erfolgt auf der Basis der Remote Shell, also des rshd (siehe S. rshd). Dementsprechend müssen für jeden Anwender eine Datei .rhosts im Heimatverzeichnis angelegt werden.

UNIX als Client

Im nächsten Schritt müssen die Daten in den Server eingecheckt werden. Dazu wechseln Sie in das Verzeichnis, in dem die Dateien stehen, die Sie in das Repository stellen möchten. Dann wird die Umgebungsvariable CVSROOT besetzt. Mit ihr wird Anwender, Rechner und Pfad festgelegt.

```
export CVSROOT=:ext:arnold@gaston:/home/cvs  
cvs import -m "Informatik-Ecke" informatik willemer BasisRe1
```

Danach stehen die Dateien allen Rechnern zur Bearbeitung zur Verfügung.

Nachdem die Daten eingecheckt sind, suchen Sie sich ein Arbeitsverzeichnis, in dem Sie an den Quellen arbeiten möchte. Hier führen Sie einen Checkout aus.

```
cd my/src  
cvs checkout informatik
```

Danach ist unter my/src ein neues Verzeichnis informatik angelegt worden. Darin finden sich die Quelldateien. Hier können Sie editieren und nach erfolgreichem Abschluss die Datei per `commit` wieder ins CVS einstellen.

```
cvs commit
```

CVS erkennt, welche Dateien sich geändert haben, verlangt für jede Änderung einen Kommentar und stellt die Änderungen zur Verfügung. Arbeiten Sie in einem Team, ist es sinnvoll, regelmäßig mit `cvs update` den aktuellsten Stand in das Arbeitsverzeichnis zu laden. Da dies die Kollegen auch tun werden, ist es erforderlich, dass ein per `commit` eingestellter Stand auch mindestens kompilierbar ist.

Windows-Client WinCVS

Auch für MS Windows gibt es einen CVS-Client, der auf einen UNIX-Server zugreift. Unter Admin - Preferences wird die CVSROOT Umgebung in einem Dialog festgelegt. Als Authentifizierung wird in diesem Fall `rhosts` gewählt. Sie können auch `ssh` wählen, wenn die Sourcen so gut kommentiert sind, dass ein Angreifer mit einem Netzlauschangriff wirklich Vorteile gewinnen könnte. Pfad, Host und User werden analog zur UNIX Umgebung festgelegt.

Sollen Windowsquellen in das Repository importiert werden, suchen Sie im linken Baum das gewünschte Verzeichnis aus und klicken es einmal an. Durch den Menüpunkt Create - Import Module oder über den rechten Mausklick mit dem entsprechenden Punkt können Sie dieses Verzeichnis für den Import wählen. Es startet ein Dialog, der den Namen des Moduls erfragt, den Vendor, die Release und eine Message.

Um die Daten auf irgendeinem Rechner bearbeiten zu können, muss zuerst ein Arbeitsverzeichnis ausgecheckt werden. Unter dem Menüpunkt Create - Checkout module können Sie angeben, welche Module lokal exportiert werden sollen. Änderungen werden dann unter Modify - Commit abgestellt.

Nachdem Dateien verändert wurden, können Sie das Modul anwählen und im Menü Modify - Commit wählen. Es werden nur die Dateien eingecheckt, die geändert worden sind.

Neue Dateien können mit Modify - Add selection oder Add binary dem Modul hinzugeführt werden.

Quelle: <http://www.cvshome.org>

Diverse Programmierhelfer

Hier sind etwas ungeordnet noch einige Werkzeuge aufgeführt, die unter UNIX ihre Heimat haben.

Unterabschnitte

- [Kurzbeschreibung: lex und yacc](#)
 - [Unterschiede zwischen Textdateien: diff](#)
 - [Dateien aufs Byte geschaut](#)
-

Kurzbetrachtung: lex und yacc

Für die Compilererstellung stehen die Programme `lex` und `yacc` zur Verfügung. Sie sollen hier nur kurz angerissen werden, da eine Beschreibung sehr schnell in die Tiefen des Compilerbaus lenken und für den großen Teil der Leser nicht sehr interessant ist.

Ein Compiler arbeitet mit mehreren Stufen. Die erste ist die lexikalische Analyse, die Schlüsselworte aus den Textdateien herausucht und erkennen kann. Der Informatiker spricht von einer regulären Sprache oder Grammatik. Das Programm `lex` generiert anhand einer Beschreibung Quelltexte, die solche Mustererkennungen durchführen können.

In der nächsten Stufe des Compilers arbeitet die syntaktische Analyse. Sie setzt auf der lexikalischen Analyse auf und interpretiert anhand einer Grammatik die Folge der von der syntaktischen Analyse gelieferten Schlüsselworte, die auch Tokens genannt werden.

Der Umgang mit `lex` und `yacc` erfordert Kenntnisse in der Theorie des Compilerbaus, die sicher nicht zu den Themengebieten dieses Buches gehören. Wenn Sie aber auf Probleme stossen, die mit dem Interpretieren von Texten im Sinne von Programmiersprachen stoßen, sollten Sie sich die beiden Programme unbedingt mit einer geeigneten Literatur Das Standardwerk zum Thema Compilerbau ist das zweibändige Werk von Aho, Alfred V./Sethi, Ravi/Ullman, Jeffrey D.: Compilerbau. Addison-Wesley, 1988. ansehen.

Informationen zu `lex` und `yacc` finden Sie auch auf folgender Webseite.

<http://www.compilerbau.de>

Unterschiede zwischen Textdateien: diff

Das Programm `diff` zeigt die Unterschiede zweier Textdateien an, die es als Parameter erwartet. Der Rückgabewert ist 0, wenn zwischen den beiden Dateien keine Unterschiede bestanden. Im folgenden Beispiel werden die Dateien `moin.c` und `doppel.c` verglichen:

```
gaston> diff moin.c doppel.c
0a1
> /* (C) 2002 Arnold willemer */
2c3
< main()
--
> main(int argc, char **argv)
5d5
<     return 0;
gaston>
```

Um den Unterschied würdigen zu können, sehen Sie hier Text von `moin.c`:

```
main()

    puts("Moin, Torfmoorholm!");
    return 0;
```

Und hier ist die Datei `doppel.c`, die an drei Stellen verändert wurde. In der ersten Zeile wurde ein Kommentar hinzugefügt. Der Funktion `main()` wurden Parameter hinzugefügt. Dafür wurde die Zeile mit `return` entfernt.

```
/* (C) 2002 Arnold willemer */

main(int argc, char **argv)

    puts("Moin, Torfmoorholm!");
```

Die mit einem Größerzeichen beginnenden Zeilen werden der Datei `moin.c` hinzugefügt, die mit dem Kleinerzeichen entfernt. Die Meldungen dazwischen geben die Zeilennummern und die Aktion an. So bedeutet a das Anhängen der Zeile 1 aus `doppel.c` hinter die Zeile 0 von `moin.c`. Der Buchstabe c bedeutet das Ersetzen und d steht für das Löschen.

Dateien aufs Byte geschaut

Sie werden es kaum glauben, aber es gibt unter UNIX auch Dateien, die nicht reine Textdateien sind. Als Programmierer muss man auch hin und wieder in solche Dateien hineinschauen.

Das Programm `od` zeigt die Datei, die ihm als Parameter übergeben wird byteweise in Oktalдарstellung an. Als Optionen können die Inhalte mit `-x` hexadezimal und mit `-c` als ASCII angezeigt werden.

```
gaston> head progr.tex | od
0000000 063534 045560 070141 072151 066145 050173 067562 071147
0000020 066541 064555 071145 067165 020147 067566 020156 064123
0000040 066145 071554 071153 070151 062564 076556 005012 061134
0000060 063545 067151 063573 040560 066156 071545 071145 005175
0000100 071120 063557 060562 066555 062551 062562 020156 071551
0000120 020164 064544 020145 067550 062550 045440 067165 072163
```

Dasselbe noch einmal in hexadezimaler Schreibweise:

```
gaston> head progr.tex | od -x
0000000 675c 4b70 7061 7469 6c65 507b 6f72 7267
0000020 6d61 696d 7265 6e75 2067 6f76 206e 6853
0000040 6c65 736c 726b 7069 6574 7d6e 0a0a 625c
0000060 6765 6e69 677b 4170 6c6e 7365 7265 0a7d
0000100 7250 676f 6172 6d6d 6569 6572 206e 7369
0000120 2074 6964 2065 6f68 6568 4b20 6e75 7473
```

Und weil es so schön ist, noch einmal als ASCII-Text:

```
gaston> head progr.tex | od -c
0000000  g  p  k  a  p  i  t  e  l  {  P  r  o  g  r
0000020  a  m  m  i  e  r  u  n  g      v  o  n      S  h
0000040  e  l  l  s  k  r  i  p  t  e  n  }  n  n      b
0000060  e  g  i  n  {  g  p  A  n  l  e  s  e  r  }  n
0000100  P  r  o  g  r  a  m  m  i  e  r  e  n      i  s
0000120  t      d  i  e      h  o  h  e      K  u  n  s  t
```

UNIX-Systemaufrufe

Programmers are a BIT smarter

Die Systemaufrufe werden von Anwendungsprogrammen benutzt, um auf die vom Betriebssystem verwalteten und damit auch geschützten Ressourcen des Computers zuzugreifen. Damit sagen die Systemaufrufe viel darüber aus, welche Leistungen ein Betriebssystem seinen Anwendern anbietet. Die Kenntnis der Systemaufrufe hilft auch beim Verständnis der Abläufe innerhalb des Systems. So wird man das Verhalten der Prozesse unter UNIX wesentlich besser verstehen, wenn man verstanden hat, wie `fork()`, `exec()` und `wait()` funktionieren.

Die Bedeutung der UNIX-Systemaufrufe geht über UNIX hinaus. Da der C-Compiler mit der Entwicklung von UNIX entstand, wurde ein Teil der Schnittstelle zu UNIX auch gleich in die Standardbibliothek von C integriert und damit als Schnittstelle für die Basiszugriffe auf andere Systeme mitgenommen. So funktionieren die Dateizugriffe auf jedem anderen Betriebssystem auch. Dagegen ist der oben genannte Aufruf von `fork()` leider nicht portabel, da andere Betriebssysteme andere Prozessmodelle haben.

-
- Die Funktion main
 - Aufrufparameter
 - Zugriff auf die Umgebungsvariablen
 - Fehlerbehandlung: `errno`
 - Dateizugriffe
 - Öffnen, Lesen und Schreiben
 - Positionieren: `lseek`
 - Dateihandle duplizieren: `dup`
 - Dateieigenschaften ermitteln
 - Dateieigenschaften ändern
 - Sperren
 - Link erzeugen: `link`, `symlink`
 - Löschen: `unlink`
 - Umbenennen: `rename`
 - Temporäre Dateien
 - Verzeichnisse
 - Auslesen: `opendir`, `readdir`, `closedir`
 - Ermitteln des Arbeitsverzeichnisses
 - Wechseln: `chdir`
 - Anlegen und Löschen: `mkdir`, `rmdir`
 - Prozesse
 - Multiprocessing contra Multithreading

- Vervielfältigen von Prozessen: fork
- exec und system
- Synchronisation: wait
- Prozessumgebung
- Gemeinsamer Speicher: Shared Memory
- Synchronisation mit Semaphoren
- Message Queues
- Leichtgewichtsprozesse: Threads
- Signale
 - Signale senden: kill
 - Auf Signale warten: pause
 - Timeout setzen: alarm
 - Zombies vereiteln
- Pipe
 - Prozesskommunikation per Pipe
 - Named Pipe oder FIFO
 - Drucken unter UNIX
- Fehlerbehandlung mit syslog
- Zeitfunktionen
- Benutzer und Gruppen
 - Die Passwortdatei als Struktur
 - Auslesen der Passwortdatei
 - Gruppen
- Grundlagen der Dämonisierung
- Client-Server Socketprogrammierung
 - Kommunikationsendpunkt: socket und close
 - Serveraufrufe: bind, listen und accept
 - Clientaufruf: connect
 - Datenaustausch: send und recv
 - Namensauflösung
 - Zahlendreher ntohs und htons
 - Rahmenprogramm eines Client-Server Paares
 - Mehrere Sockets parallel abfragen
 - IPv6 aus Programmiersicht
 - Client-Server aus Sicht der Performance
- Reguläre Ausdrücke
- Weitere Programmierschnittstellen
- Systemkonformität
 - Polling
 - Rechte beachten

Die Funktion main

Bevor auf die Systemaufrufe eingegangen wird, soll die Umgebung einer normalen Anwendung in UNIX betrachtet werden. Dazu gehören Aufrufparameter, Umgebungsvariablen und Rückgabewert. Ein C-Programm wird immer mit der Funktion `main()` gestartet. Diese hat drei Parameter, von denen in den meisten Fällen nur die ersten beiden verwendet werden. Der dritte Parameter hält das Environment, das aber auch per `getenv()` gelesen werden kann. Er wird von POSIX nicht mehr unterstützt.

```
int main(int argc, char **argv)
{
    return 0;
}
```

Unterabschnitte

- [Aufrufparameter](#)
 - [Zugriff auf die Umgebungsvariablen](#)
-

Aufrufparameter

In der Variablen `argc` steht nach dem Start, wieviele Parameter bei Aufruf des Programmes übergeben wurden. Dieser Wert ist immer mindestens 1, da auch der Name, unter dem das Programm aufgerufen wurde, als Parameter zählt. Die Variable `argv` ist ein Array von Strings. Die Programmparameter werden von der Shell an den Leerzeichen zerlegt und dann einzeln an das Programm weitergereicht. Beispiel:

```
tudochwas -f huhu lol*
```

Der Wildcard von `lol*` wird bekanntermaßen von der Shell ausgewertet. Wenn im aktuellen Verzeichnis die Dateien `lolita`, `lolli` und `lonzo` stehen, sind die Parameter von `main()` folgendermaßen belegt:

In `argc` steht eine 5. Im Element `argv[0]` findet sich der Name des Programmes inklusive Pfad, in diesem Fall `./tudochwas`. Da `tudochwas` aus dem aktuellen Verzeichnis gestartet wurde und kein Pfad angegeben wurde, bemühte die Shell die Variable `PATH`. Darin fand sie dann den Pfad `.` und setzte ihn vor den Programmnamen. `argv[1]` enthält die Option `-f` und `argv[2]` den Parameter »huhu«. Die weiteren Elemente von `argv` hängen von den Dateien im aktuellen Verzeichnis ab. Da auf `lol*` sowohl `lolita` als auch `lolli` passen, füllen diese die nächsten zwei `argv`-Elemente.

Da das erste Element in `argv` immer der Name ist, mit dem das Programm aufgerufen wurde, kann man dem Programm Optionen durch den Befehlsnamen mitgeben. So entspricht beispielsweise der Befehl `uncompress` dem Aufruf `compress -d`. Das erreichen Sie, indem ein Link namens `uncompress` auf `compress` erzeugen. So ein Link kostet unter UNIX nur einen Verzeichniseintrag mehr. Das Programm prüft beim Start, ob es unter dem Namen `uncompress` aufgerufen wurde und fügt in diesem Fall die Option `-d` hinzu.

An den letzten Argumenten erkennt man, dass der Stern unter UNIX von der Shell aufgelöst wird. Dadurch ist erreicht, dass alle Programme von Haus aus die Wildcards der Shell gleichermaßen interpretieren, da sie sie selbst nicht auswerten.

Das folgende kleine Programm zeigt seine Aufrufparameter an:

```
[Zeigt die Aufrufparameter]
int main(int argc, char **argv)
{
    int i;
    for (i=0; i<argc; i++) {
        puts(argv[i]);
    }
    return 0;
}
```

Auch der Rückgabewert der Funktion `main()` ist von Bedeutung, da dieser an den Aufrufer zurückgegeben wird. Als Konvention gilt eine 0 als Hinweis, dass das Programm fehlerfreier ablief. Alle anderen Werte werden als Fehlermeldungen interpretiert. Wenn Fehler mitten im Programm entstehen, ist es oft sehr umständlich, wieder zur Funktion `main()` zurückzukehren. Hier hilft die Funktion `exit()`.

Sie beendet das Programm und der Übergabeparameter wird als Rückgabewert an den Aufrufer des Programms weitergereicht.

« [Die Funktion main](#) | **Die Funktion main** | [Zugriff auf die Umgebungsvariablen](#) »

Zugriff auf die Umgebungsvariablen

Der bislang nicht erwähnte dritte Parameter von `main()` zeigt auf die Umgebungsvariablenliste. Wie bei `argv` handelt es sich um ein Array von Zeichenketten. Da hier allerdings die Anzahl nicht in einer separaten Variablen mitgeteilt wird, wird eine andere Form der Endekennung verwendet. Der Zeiger nach dem letzten Eintrag hat den Wert 0. In dem Beispielprogramm wird die komplette Liste der Umgebungsvariablen angezeigt.

```
[Zeigt die Liste der Umgebungsvariablen]
main(int argc, char **argv, char **env)
{
    int i;
    char *str;
```

```
    if (env) {
        i=0;
        while(str = env[i]) {
            puts(env[i++]);
        }
    }
}
```

Wer das Programm startet bekommt eine riesige Liste, von der hier nur ein kurzer Ausschnitt angezeigt wird:

```
WINDOWMANAGER=/usr/X11R6/bin/kde
HOME=/home/arnold
TERM=xterm
XNLSPATH=/usr/X11R6/lib/X11/nls
no_proxy=localhost
```

Der dritte Parameter ist deswegen relativ unbekannt, weil man über die externe, globale Variable `environ` auf die gleiche Liste zugreifen kann. Aus diesem Grund hat sich POSIX auch auf die Variante mit zwei Parametern festgelegt.

Im Normalfall werden Sie allerdings mit der kompletten Liste wenig anfangen wollen, sondern werden eine spezielle Umgebungsvariable brauchen. Dazu verwenden Sie die Funktion `getenv()`.

```
#include <stdlib.h>
char *getenv(const char *varname);
int putenv(const char *zuweisung);
```

Sofern die als Parameter angegebene Variable gesetzt ist, bekommen Sie einen Zeiger auf den Wert zurückgeliefert. Da dieser Zeiger beim nächsten Aufruf von `getenv()` verloren gehen kann, sollten Sie den Inhalt in eine lokale Variable kopieren, wenn Sie ihn später noch brauchen.

Es ist auch möglich, vom Programm aus Umgebungsvariablen zu setzen. Dazu dient die Funktion `setenv()`. Als Parameter erwartet sie einen vollständigen Variableneintrag der Form `var=wert`. Beispiel:

```
putenv("TERM=vt100");
```

« [Aufrufparameter](#) | **Die Funktion main** | [Fehlerbehandlung: errno](#) »

Fehlerbehandlung: `errno`

Die meisten Systemaufrufe liefern einen Rückgabewert kleiner 0, wenn etwas schief gelaufen ist. Ist der Rückgabewert nicht aussagekräftig genug, so verwendet UNIX die globale Variable `errno`. Hier findet sich der Grund für das Fehlschlagen. Die Konstanten, die `errno` annehmen kann, stehen in der Headerdatei `errno.h` zur Verfügung.

Die Funktion `perror()` gibt die Standardfehlermeldungen des Betriebssystems auf `stderr` aus. Die Anwendung kann `perror()` im Parameter einen Text mitgeben, der erläutert, in welchem Zusammenhang der Fehler auftritt. Dieser Text wird der Systemmeldung vorangestellt.

```
#include <stdio.h>
void perror(const char *meldung);
```

Da es sich beispielsweise bei X-Anwendungen gut macht, wenn die Fehlermeldung nicht auf `stderr`, sondern beispielsweise in einer Dialogbox erscheint, verwendet man dort statt des Aufrufes von `perror()` die Funktion `strerror(errno)`. Sie liefert den Meldungstext passend zur Fehlermeldung der Anwendung als Zeichenkette, die das Programm dann an beliebiger Stelle darstellen kann.

```
#include <string.h>
char *strerror(int errnum);
```

Dateizugriffe

UNIX hat eine Reihe von Standarddateizugriffen definiert, die mit C auch auf die anderen Plattformen übertragen wurden. Nirgends ist aber der Dateibegriff so universell wie unter UNIX. Mit den gleichen Funktionen können Dateien, Pipes und sogar Netzverbindungen bearbeitet werden.

Neben den Dateizugriffen, die durch die Aufrufe `open()`, `read()`, `write()` und `close()` geprägt sind und die einen Integer, also eine ganze Zahl, als Dateihandle zurückgeben, gibt es noch die Dateizugriffe mit `fopen()`, `fread()`, `fwrite()` und `fclose()`, die als Dateihandle einen Zeiger auf die Struktur `FILE` zurückgeben. Diese in der Anwendungsprogrammierung weiter verbreitete Variante basiert auf den zuerst genannten Basiszugriffen und erweitert sie um ein Pufferkonzept. Sie gehören auch nicht zu den Systemaufrufen, die vom Betriebssystem behandelt werden, sondern sind Bibliotheksfunktionen. Nähere Informationen dazu finden sich in den Dokumentationen der Compiler oder natürlich unter `man 3 fopen`.

Unterabschnitte

- [Öffnen, Lesen und Schreiben](#)
 - [Öffnen: open](#)
 - [Schließen: close](#)
 - [Lesen: read](#)
 - [Schreiben: write](#)
- [Positionieren: lseek](#)
- [Dateihandle duplizieren: dup](#)
- [Dateieigenschaften ermitteln](#)
 - [stat\(\) und fstat\(\)](#)
 - [Zugriffsrecht ermitteln](#)
- [Dateieigenschaften ändern](#)
 - [Zugriffsrechte ändern](#)
 - [Besitzer und Gruppe ändern](#)
 - [Maske für die Rechte neuer Dateien: umask](#)
- [Sperren](#)
 - [Sperren nach POSIX](#)
 - [lockf](#)
 - [flock](#)
 - [locking](#)
 - [advisory und mandatory](#)
- [Link erzeugen: link, symlink](#)
- [Löschen: unlink](#)
- [Umbenennen: rename](#)

- Temporäre Dateien

« Fehlerbehandlung: errno | **UNIX-Systemaufrufe** | Öffnen, Lesen und Schreiben »

Unterabschnitte

- Öffnen: [open](#)
- Schließen: [close](#)
- Lesen: [read](#)
- Schreiben: [write](#)

Öffnen, Lesen und Schreiben

Um eine Datei lesen oder schreiben zu können, muss sie zunächst durch den Aufruf von `open()` geöffnet werden. Vor Ende des Programmes wird sie mit `close()` wieder geschlossen.

Öffnen: `open`

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(char *dateiname, int oflag, mode_t modus);
```

Die Parameter haben folgende Bedeutung:

- [dateiname] Der Pfad und Dateiname der zu öffnenden Datei.
- [oflag] Dieser Parameter bestimmt, wie die Datei geöffnet wird. Es können mehrere Attribute verwendet werden, indem sie mit einander durch den senkrechten Strich mit Oder verknüpft werden.

[Dateiattribute bei `open()`]|L|L Konstante & Bedeutung

`O_RDONLY` & Datei nur zum Lesen öffnen

`O_WRONLY` & Datei nur zum Schreiben öffnen

`O_RDWR` & Datei zum Lesen und Schreiben öffnen

`O_APPEND` & Es wird ans Ende der Datei angehängt

`O_CREAT` & Erzeuge die Datei, wenn sie nicht existiert

`O_EXCL` & Exklusiver Zugriff

- [modus] Hier werden Schreib- und Leserechte angegeben, wie sie vom `chmod` bekannt sind.

Bei einem Fehler gibt `open()` -1 zurück. Der Fehlercode befindet sich in der globalen Variablen `errno`. Im Erfolgsfall wird ein Dateihandle vom Typ `int` zurückgegeben, der von den Dateifunktionen benötigt wird, um die entsprechende Datei weiter zu bearbeiten.

Das Flag `O_APPEND` ist vor allem beim Beschreiben von Protokolldateien wichtig. Beim Schreiben werden die Daten immer hinten an die Datei angehängt.

In einer Multitaskingumgebung wie UNIX ist diese Art der Positionierung besonders wichtig. Statt zunächst die Größe der Datei und daraus die Schreibposition zu ermitteln, reicht ein einzelner Aufruf von `write()` um zu schreiben. Damit ist die Operation untrennbar. Untrennbar bedeutet, dass die Operation beendet wird, bevor ein anderer Prozess dieselbe Operation ausführen kann. Ein paralleler

Prozess, der auch schreiben will, kann also nur davor oder dahinter schreiben, aber nicht an die gleiche Stelle. Dagegen ist der scheinbar gleichwertige Ansatz mit `lseek()` und `write()` ohne `O_APPEND` als Parameter für `open()` teilbar, da es zwei Operationen sind. Dann können parallele Prozesse versehentlich in den gleichen Bereich schreiben.

Die Kombination `O_CREAT | O_EXCL` ist wunderbar zur Synchronisation mehrerer Prozesse verwendbar. Da ein solcher `open()` ein untrennbarer Aufruf ist, kann immer nur ein Prozess die Datei im Zugriff haben. Beispielsweise können Sie diese Technik verwenden, um zu vermeiden, dass zwei Prozesse gleichzeitig in einem kritischen Bereich arbeiten. Dazu wird zuvor versucht, eine Datei, die beispielsweise `lock` heißen kann, mit den oben angegebenen Parameter zu erzeugen. Der Aufruf misslingt, wenn bereits eine `lock`-Datei existiert. Derjenige, der also diesen Aufruf erfolgreich durchführen kann, ist also der einzige Prozess im kritischen Bereich. Nach Ende der Arbeiten löscht er einfach wieder die Datei `lock` und läutet damit die nächste Runde ein.

Statt dem Aufruf `open()` mit der Option `O_CREAT` können Sie auch `creat()` verwenden.

```
int creat(const char *dateiname, mode_t modus);
```

Zum Schließen der Datei wird der Aufruf `close()` verwendet.

Schließen: close

Der Aufruf von `close()` schließt die Datei wieder.

```
int close(int dateihandle);
```

Dass eine geöffnete Datei auch wieder geschlossen werden sollte, sagt Ihnen bereits Ihr Ordnungssinn. Darüber hinaus ist es einsehbar, dass bei vielen geöffneten Dateien der Verwaltungsaufwand für das Betriebssystem steigt. Der Hauptgrund, eine Datei möglichst bald wieder zu schließen, ist aber der, dass eine offene Datei immer ein Risiko darstellt, da der Zustand der Datei in der Schwebe ist.

Als Parameter wird `close()` das Dateihandle übergeben.

Der Rückgabewert ist 0, wenn alles in Ordnung ist und -1, wenn ein Fehler auftritt. In bestimmten Fällen kann ein Fehler in einer vom System gepufferten Schreiboperation erst beim Schließen der Datei auffallen. Insofern sollten Sie auch den Rückgabewert von `close()` prüfen, wenn Sie eine sichere Information brauchen, ob die Dateioperationen geklappt haben.

Lesen: read

Der Aufruf `read()` liest Daten aus einer geöffneten Datei.

```
#include <unistd.h>
int read(int dateihandle, void *puffer, size_t laenge);
```

Damit aus einer Datei gelesen werden kann, muss sie geöffnet sein. Den Rückgabewert von `open()` braucht `read()` als ersten Parameter. Vor dem Lesen muss ein Puffer angelegt werden, in dem die zu

lesenden Daten abgelegt werden. Die Adresse dieses Speichers wird als weiterer Parameter benötigt. Schließlich muss angegeben werden, wie groß der Datenblock sein soll, der gelesen wird. Dieser Parameter sollte nicht zu klein gewählt werden. Das byteweise Einlesen einer mittelgroßen Datei kann ein Programm minutenlang beschäftigen.

Der Rückgabewert gibt an, wieviele Bytes gelesen wurden und er sollte mit dem Parameter `laenge` übereinstimmen. Ist er kleiner, vermutet man leicht als Grund das Ende der Datei bzw. der Sendung. Da aber beispielsweise im Netz Verzögerungen auftreten können, sollten Sie auch in diesem Fall noch einmal lesen, bis der Rückgabewert 0 ist. Ist der Rückgabewert -1, ist ein Fehler aufgetreten. Nähere Informationen zu der Ursache des Fehlers finden Sie in der Variablen `errno`.

Schreiben: `write`

Mit dem Aufruf `write()` können Sie in eine geöffnete Datei schreiben.

```
#include <unistd.h>
int write(int dateihandle, void *puffer, size_t laenge);
```

Auch `write()` braucht das Dateihandle als ersten Parameter. Der zweite Parameter gibt die Adresse des Speichers an, aus dem geschrieben werden soll. Im letzten Parameter wird angegeben, wie viele Zeichen geschrieben werden sollen.

Der Rückgabewert ist wie bei `read()` normalerweise identisch mit dem Parameter `laenge`. Im Fehlerfall ist er -1 und die Variable `errno` gibt über die Ursache Auskunft.

« [Dateizugriffe](#) | [Dateizugriffe](#) | [Positionieren: lseek](#) »

Positionieren: lseek

Ein Lesezugriff liest einen Block aus einer bereits geöffneten Datei. Er fängt normalerweise vorn an und setzt an der zuletzt erreichten Stelle wieder auf, um einen weiteren Lesezugriff zu bedienen. Dasselbe geschieht beim Schreibzugriff. Es gibt also einen Dateipositionszeiger, der immer an die Stelle der nächsten Dateioperation zeigt.

Wollen Sie innerhalb der Datei an einer bestimmten Stelle anfangen zu lesen, können Sie mit dem Aufruf `lseek()` dorthin springen.

```
int lseek(int dateihandle, off_t offset, int woher);
```

Der Parameter `offset` bestimmt die Position, ab der die nächste Operation beginnt. Der Parameter `woher` legt fest, von wo der Offset berechnet wird.

[lseek Parameter Woher]L|L Konstante & Bedeutung
SEEK_SET & ab Dateianfang
SEEK_CUR & ab der aktuellen Position
SEEK_END & ab Dateiende

Mit dem Aufruf `lseek()` ist es möglich, Löcher in Dateien entstehen zu lassen. Wird beispielsweise bei einer Datei mit einem Byte Länge an die Position 1000 gesprungen und geschrieben, gibt es ein Datenloch, das nicht auf der Platte angelegt wird. Für das Programm bleibt das transparent. Lesen Sie später einen Bereich aus diesem Loch, erhalten Sie einen mit Nullen aufgefüllten Speicherbereich. Problematischer ist es, wenn Sie später sukzessive das Loch beschreiben, da die Speicherbereiche auf der Platte dann nicht mehr linear vorliegen und mit Einbußen bei der Performance zu rechnen ist.

Dateihandle duplizieren: dup

Werden zwei Dateihandle für dieselbe Datei gebraucht, ruft man dup() auf.

```
#include <unistd.h>
int dup(int altdateihandle);
```

Diese Funktion gibt ein Dateihandle zurück, der auf die gleiche Datei zeigt. Auf diese Weise ist es beispielsweise möglich an zwei verschiedenen Stellen gleichzeitig in einer Datei zu arbeiten, da jedes Handle seinen eigenen Positionszeiger besitzt. Ist es notwendig, dass das Handle eine bestimmte Nummer hat, benötigen Sie dup2.

```
#include <unistd.h>
int dup2(int altdateihandle, int doppelhandle);
```

Unterabschnitte

- [stat\(\) und fstat\(\)](#)
- [Zugriffsrecht ermitteln](#)

Dateieigenschaften ermitteln

Sie können aus dem Programm heraus ermitteln, wie die Zugriffsrechte auf eine Datei sind. Sie erhalten Informationen über Zeiten des letzten Schreibens und Lesens. Sie können den Besitzer und den Typ der Datei feststellen.

stat() und fstat()

Mit der Funktion `stat()` und `fstat()` können Sie Informationen über eine Datei ermitteln. Die Ergebnisse werden in einer Struktur vom Typ `stat` abgelegt. Strukturen sind zusammengesetzte Typen. Sie müssen eine Variable dieses Typs anlegen und dessen Adresse der Funktion `stat()` übergeben. Der Unterschied zwischen beiden Funktionen liegt darin, dass `stat()` den Dateinamen und `fstat()` das Dateihandle zur Identifikation der Datei verwendet.

```
#include <sys/types.h>
#include <sys/stat.h>
int stat(char *dateiname, struct stat *puffer);
int fstat(int dateihandle, struct stat *puffer);
```

Die Ergebnisse finden sich in der Variablen von Typ `stat`, auf die der Parameter `puffer` zeigt. Die Definition der Struktur `stat` lautet:

```
struct stat {
    dev_t  st_dev      /* (P) Device, auf dem die Datei liegt */
    ushort st_ino      /* (P) Inode-Nummer */
    ushort st_mode     /* (P) Dateityp */
    short  st_nlink     /* (P) Anzahl der Links der Datei */
    ushort st_uid      /* (P) Eigner-ID (uid) */
    ushort st_gid      /* (P) Gruppen-ID (gid) */
    dev_t  st_rdev     /* Major- und Minornumber, falls Device */
    off_t  st_size     /* (P) Größe in Byte */
    time_t st_atime     /* (P) Zeitpunkt letzter Zugriffs */
    time_t st_mtime     /* (P) Zeitpunkt letzte Änderung */
    time_t st_ctime     /* (P) Zeitpunkt letzte Statusänderung */
};
```

Die Bestandteile dieser Struktur können sich je nach System unterscheiden. Die mit (P) gekennzeichneten Elemente sind aber zwingend von POSIX vorgeschrieben.

`st_dev` und `st_ino` beschreiben eindeutig den Ort einer Datei. `st_dev` ist das Device, bei Festplatten also die Partition. `st_ino` bezeichnet den i-node, in dem auf die Datei verwiesen wird.

Die rechten 12 Bit von `st_mode` beschreiben die Rechtezuordnung der Datei, wie sie von `chmod` (siehe S. `chmod`) bekannt ist. Zu berücksichtigen ist, dass die Werte oktal sind, Der Modus 755 muss also als 0755 in einem C-Programm dargestellt werden. In den nächsten vier Bit wird kodiert, welchen Typ die Datei hat. Um beides zu trennen, gibt es die Konstante `S_IFMT`. Mit ihr können Sie eine Maske über diese Bits setzen. Anschließend können Sie den Wert mit folgenden Konstanten vergleichen:

[Konstanten für den Dateityp]L|L Konstante & Dateityp

`S_IFSOCK` & Sockets

`S_IFLNK` & Symbolische Links

`S_IFREG` & reguläre Dateien

`S_IFBLK` & Block-Devices

`S_IFDIR` & Verzeichnisse

`S_IFCHR` & Char-Devices

`S_FIFO` & FIFOs

Das folgende Beispielprogramm ermittelt für die als ersten Parameter übergebene Dateinamen die Rechte und stellt anschließend fest, ob es sich um eine Datei, einen symbolischen Link oder ein Verzeichnis handelt.

[Dateityp und Rechte bestimmen]

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
int main(int argc, char **argv)
```

```
{
    struct stat Status;
```

```
    int Dateityp;
```

```
    stat(argv[1], &Status);
```

```
    /* Dateirechte */
```

```
    printf("Dateirechte: %o n", Status.st_mode & S_IFMT);
```

```
    Dateityp = Status.st_mode & S_IFMT;
```

```
    switch (Dateityp) {
```

```
        case S_IFREG: puts("Datei"); break;
```

```
        case S_IFLNK: puts("Symbolischer Link"); break;
```

```
        case S_IFDIR: puts("Verzeichnis"); break;
```

```
        default: puts("Sonstiges");
```

```
    }
```

```
}
```

In `st_nlink` steht, wieviele harte Links auf die Datei zeigen.

Mit `st_uid` und `st_gid` werden der Besitzer und die Besitzergruppe ermittelt. Der Wert ist eine Zahl, nämlich die, welche in `/etc/passwd` bzw. in `/etc/group` festgelegt wird.

In `st_rdev` ist die Major- und Minornummer kodiert, sofern es sich bei der Datei um ein Device handelt.

Sofern es sich bei der Datei um eine reguläre Datei handelt, finden Sie in `st_size` die Größe in Bytes.

Jeder lesende oder schreibende Zugriff auf die Datei aktualisiert den Wert `st_atime`. Jede Veränderung des Dateiinhalts wird in `st_mtime` notiert. Der Zeitpunkt der Änderungen bezüglich der Benutzer, Rechte, Linkzahl oder ähnlichem, also allem, was nicht den Inhalt betrifft, wird in `st_ctime` festgehalten.

Zugriffsrecht ermitteln

Will ein Programm ermitteln, ob es beispielsweise Schreibrecht auf eine Datei hat, könnte es mit `stat` alle Informationen über die Datei auswerten. Schneller geht es mit der Funktion `access()`, die nur ermittelt, ob das gewünschte Recht verfügbar ist.

```
#include <unistd.h>
int access(const char *dateiname, int modus);
```

An den Parameter `modus` können folgende Konstanten übergeben werden:

[Rechte beim Aufruf von `access`] L|L Konstante & Bedeutung

F_OK & Prüfung auf Existenz der Datei

R_OK & Prozess darf lesen

W_OK & Prozess darf schreiben

X_OK & Prozess darf die Datei ausführen

Der Rückgabewert ist 0, wenn der Zugriff erlaubt ist oder EACCESS, wenn nicht.

« [Dateihandle duplizieren: dup](#) | [Dateizugriffe](#) | [Dateieigenschaften ändern](#) »

Unterabschnitte

- [Zugriffsrechte ändern](#)
- [Besitzer und Gruppe ändern](#)
- [Maske für die Rechte neuer Dateien: umask](#)

Dateieigenschaften ändern

Die UNIX-Kommandos `chmod`, `chown` und `chgrp` basieren natürlich auf Systemaufrufen, die sich auch aus einem Programm heraus aufrufen lassen.

Zugriffsrechte ändern

Das Kommando `chmod` (siehe S. `chmod`) hat sein direktes Gegenstück in der Funktion `chmod()` bzw. `fchmod()`. Beide Funktionen unterscheiden sich darin, ob die Datei durch ein Dateihandle oder durch ihren Dateinamen bestimmt wird.

```
#include <sys/stat.h>
int chmod(const char *dateiname, mode_t modus);
int fchmod(int dateihandle, mode_t modus);
```

Der Parameter `modus` erhält die Kodierung der Rechte, wie man sie vom Kommando `chmod` kennt. Dem Parameter des Kommandozeilenwerkzeug wird eine Null vorangesetzt, damit der Wert oktal übergeben wird. Würde Sie auf der Kommandozeile 644 verwenden, müssten Sie im Programm `chmod(fh, 0644)` verwenden.

Besitzer und Gruppe ändern

Auch für die Aufrufe `chown` und `chgrp` gibt es Gegenstücke in den Systemaufrufen. Allerdings werden beide Funktionalitäten durch die gleiche Funktion `chown()` bzw. `fchown()` behandelt. Eine Funktion `chgrp()` gibt es demzufolge nicht.

```
#include <sys/stat.h>
int chown(char *dateiname, uid_t userID, gid_t groupID);
int fchown(int dateihandle, uid_t userID, gid_t groupID);
```

Mit diesen Funktionen kann Benutzer und Gruppe einer Datei geändert werden. Wollen Sie nur den Wert des Benutzers oder nur der Gruppe ändern, wird für den jeweils anderen Parameter `-1` übergeben.

Maske für die Rechte neuer Dateien: umask

Diese Maske legt fest, welche Berechtigungen vom Programm bei der Erzeugung einer Datei oder Verzeichnisses nicht vergeben wird. Die Berechtigungen sind wie bei `chmod` aufgebaut, allerdings

negiert.

```
#include <sys/stat.h>
int umask(int maske);
```

Der Rückgabewert ist die vormals geltende Maske.

« [Dateieigenschaften ermitteln](#) | [Dateizugriffe](#) | [Sperren](#) »

Unterabschnitte

- [Sperren nach POSIX](#)
- [lockf](#)
- [flock](#)
- [locking](#)
- [advisory und mandatory](#)

Sperren

Wenn mehrere Prozesse in ein und derselben Datei Änderungen vornehmen wollen, wird es notwendig, Bereiche dieser Datei zu sperren. Diese Problematik ist bei den ersten Versionen von UNIX nicht berücksichtigt worden. Erst später beim Einsatz im kommerziellen Umfeld wurde die Notwendigkeit erkannt und in den verschiedenen UNIX-Dialekten unterschiedlich nachgereicht. So gibt es mehrere API-Aufrufe, um einen Dateibereich zu sperren. Relevant ist der Standard unter POSIX, den Sie bei Wahlmöglichkeit auch einsetzen sollten. Es kann aber durchaus passieren, dass Sie auf einen der anderen Sperrmechanismen in älteren Programmen stoßen, die aus Kompatibilitätsgründen oft noch funktionieren.

Nicht jedes Dateisystem unterstützt Sperren. Insbesondere wenn es sich um ein Netzdateisystem handelt, kann es Schwierigkeiten geben. Aus diesem Grund sollte das Funktionieren der Sperren vor dem Einsatz in der Produktionsumgebung getestet werden.

Sperren nach POSIX

Die POSIX-Variante verwendet zum Sperren von Dateiausschnitten die Funktion `fcntl()`.

```
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
fcntl(int dateihandle, int kommando, struct flock *sperre);
```

Der Parameter `kommando` gibt die Funktion an und kann folgende Werte annehmen:

[Sperrfunktionalität]L|L Konstante & Bedeutung
F_GETLK & Ermittle, ob eine Sperre gesetzt ist
F_SETLK & Eine Sperre setzen oder Fehler zurückgeben
F_SETLKW & Eine Sperre setzen oder warten

Die Elemente der Struktur `flock` geben die näheren Informationen über die Sperre an.

[Struktur flock]L|L|L Element & Wert & Bedeutung
& F_WRLCK & exklusiv
l_type & F_RDLCK & geteilt, verhindere Schreibzugriff

& F_UNLCK & Sperre aufheben

& SEEK_SET & vom Dateianfang

l_whence & SEEK_CUR & ab aktueller Dateiposition

& SEEK_END & vom Dateiende

l_start & Offset des Sperrbereichs

l_len & Länge des Bereichs, bei 0 bis Dateiende

Das flock-Element l_pid liefert bei F_GETLK den Prozess, der die Sperre gesetzt hat.

Als Beispiele dienen zwei Programme, die von zwei Terminalsitzungen aus gestartet werden. Das eine heißt one.c und das andere two.c. Zentraler Bestandteil der Programme ist die Funktion Sperren().

```
[Sperrfunktion Version 1]
/* Sperren() aus one.c */
#include <unistd.h>
#include <stdio.h>
```

```
int Sperren(char *name)
{
    long data;
    int fh;
    struct flock sperre;
```

```
fh = open(name, O_RDWR, 0644);
    if (0 < fh) {
        sperre.l_type = F_WRLCK;
        sperre.l_whence = SEEK_SET;
        sperre.l_start = 5 * sizeof(data);
        sperre.l_len = sizeof(data);
        if (0 > fcntl(fh, F_SETLK, &sperre)) {
            perror("Versuche die erste Sperre");
        }
        getchar(); /* nun ist der Satz gesperrt */
        sperre.l_type = F_UNLCK;
        if (0 > fcntl(fh, F_SETLK, &sperre)) {
            perror("Löse die erste Sperre");
        }
        close(fh);
    } else {
        perror("kann Datei nicht öffnen");
    }
}
```

Das Programm one wird gestartet und bleibt in der Funktion Sperren() an der Stelle stehen, wo es mit getchar() auf die Return Taste wartet. In dieser Zeit wird auf der zweiten Konsole das Programm two gestartet. Dessen Funktion Sperren() sieht nur geringfügig anders aus:

```
[Sperrfunktion Version 2]
int Sperren(char *name)
{
    long data;
    int fh;
    struct flock sperre;
```

```

fh = open(name, O_RDWR, 0644);
if (0 < fh) {
    puts("two: vor lock");
    sperre.l_type = F_WRLCK;
    sperre.l_whence = SEEK_SET;
    sperre.l_start = 5 * sizeof(data);
    sperre.l_len = sizeof(data);
    if (0 > fcntl(fh, F_SETLKW, &sperre)) {
        perror("Versuche die zweite Sperre");
    }
    puts("two: nach lock");
    getchar();
    sperre.l_type = F_UNLCK;
    if (0 > fcntl(fh, F_SETLKW, &sperre)) {
        perror("Löse die zweite Sperre");
    }
    close(fh);
}
}

```

Das Programm two wird »two: vor lock« ausgeben und dann stehen. Es blockiert also an der von one gesetzten Sperre. Wird das Programm one durch Drücken der Return-Taste weitergeführt, gibt es die Sperre frei und two meldet sich mit »two: nach lock«. Nun hält Programm two die Sperre. Starten Sie jetzt das Programm one, erhalten Sie die folgende Meldung:

```
Versuche die erste Sperre: Resource temporarily unavailable
```

Ganz offensichtlich blockiert one nicht, sondern bekommt eine Fehlerückgabe durch `fcntl()`. Das hängt damit zusammen, dass die Sperre im Programm one.c mit dem Kommando `F_SETLK` angefordert wird, die nicht blockiert, wenn die Sperre gesetzt ist, sondern eine Fehlermeldung liefert. Dagegen arbeitet two.c mit dem Kommando `F_SETLKW`. Dann blockiert der Prozess, bis die Sperre wieder freigegeben wird.

Zuletzt soll bei einer bestehenden Sperre ermittelt werden, welcher Prozess die Sperre setzt. Dazu verwenden wir die Funktion `Sniff()`:

```

[Blockade suchen]
int Sniff(char *name)
{
    long data;
    int fh;
    struct flock sperre;

```

```

fh = open(name, O_RDWR, 0644);
if (0 < fh) {
    sperre.l_type = F_WRLCK;
    sperre.l_whence = SEEK_SET;
    sperre.l_start = 5 * sizeof(data);
    sperre.l_len = sizeof(data);
    if (0 > fcntl(fh, F_GETLK, &sperre)) {
        perror("Problem bei fcntl");
    }
    printf("Prozess: %dn", sperre.l_pid);
    close(fh);
}
}

```

lockf

Die Funktion `lockf` wurde durch System V eingeführt. Sie wird von manchen Systemen, beispielsweise Linux, als weitere Schnittstelle zu den POSIX-Sperren akzeptiert. Neue Entwicklungen sollte man dennoch auf POSIX ausrichten.

```
#include <sys/file.h>
int lockf(int dateihandle, int kommando, off_t laenge);
```

Der Parameter `kommando` wird mit einer Konstanten besetzt, die die Funktion angibt.

- `[F_LOCK]` Setzt eine exklusive Sperre. Ist der Bereich bereits gesperrt, blockiert der Prozess. So ist gewährleistet, dass nur ein Prozess eine Sperre setzen kann.
- `[F_TLOCK]` Setzt wie `F_LOCK` eine exklusive Sperre. Bei einer bereits vorliegenden Sperre blockiert der Prozess aber nicht. Statt dessen gibt die Funktion einen Fehler zurück.
- `[F_ULOCK]` Hebt die Sperre auf.
- `[F_TEST]` Prüft, ob eine Sperre auf diesem Bereich liegt. Die Funktion gibt -1, wenn ein anderer Prozess die Sperre gesetzt hat. Der Rückgabewert 0 bedeutet, dass keine Sperre vorliegt oder die Sperre vom eigenen Prozess stammt.

Mit dem Parameter `laenge` kann die Länge des zu sperrenden Bereichs festgelegt werden. Die Position innerhalb der Datei wird durch einen vorangehenden `lseek()` bestimmt.

flock

Diese Funktion zum Sperren von ganzen Datei wurde durch BSD 4.2 eingeführt. `flock` sperrt nicht einen Ausschnitt, sondern die komplette Datei.

```
flock(int dateihandle, int operation)
```

Der Parameter `operation` kann folgende Werte annehmen.

- `[LOCK_SH]` Shared lock. Mehrere Prozesse können parallel sperren. Dies ist eine eher ungewöhnliche Operation und macht nur im Zusammenhang mit mandantory locking (siehe unten) Sinn.
- `[LOCK_EX]` Exclusive lock. Nur ein Prozess kann sperren.
- `[LOCK_UN]` Unlock. Die Sperre wird aufgehoben.
- `[LOCK_NB]` Nonblocking . Die Operation blockiert nicht, sondern gibt eine Fehlermeldung zurück. Da dies keine eigene Operation ist, muss sie mit einer der oben beschriebenen Operation per Oder (also mit einem einzelnen senkrechten Strich) verknüpft werden.

Die Funktion gibt im Erfolgsfall 0 ansonsten -1 zurück. Nähere Informationen finden Sie dann in der Variablen `errno`.

locking

Die Funktion `locking` wurde unter XENIX 3vgl. Rochkind, Marc J.: UNIX Programmierung für Fortgeschrittene. Hanser, München-Wien, 1988. S. 256-261. eingeführt.

```
int locking(int dateihandle, int kommando, long laenge);
```

Der Parameter `kommando` kann folgende Werte annehmen.

[Sperroperation bei locking]L|L Konstante & Bedeutung

LK_LOCK & Eine Sperre setzen

LK_UNLOCK & Eine Sperre aufheben

Die Sperre wird ab der aktuellen Dateiposition gesetzt. Als eigenen Parameter kennt locking nur die Länge des zu sperrenden Bereichs. Um die Dateiposition zu ändern, müssen Sie zuvor lseek() aufrufen.

advisory und mandatory

Es gibt zwei Arten des Sperrens: advisory (übersetzt etwa empfohlen) und mandatory (übersetzt zwingend). Beim advisory locking wird der gesperrte Bereich nur dadurch geschützt, dass alle Programme, die auf die Datei zugreifen, nur über die Sperrmechanismen zugreifen. Greift ein anderer Prozess zu oder hält sich jemand nicht an diese Abmachung, hat die Sperre keinen Wert.

Beim Aktivieren des mandatory locking wird die Sperre durch das System überwacht. Alle Dateizugriffe werden geprüft, um die Sperre zu schützen. Mandatory locking wird mit denselben Aufrufen realisiert wie das advisory locking. Der Umstieg erfolgt durch das Setzen der Dateiattribute. Mit dem chmod wird das Set-Group-ID-Bit gesetzt und die Ausführbarkeit für die Gruppe gelöscht (siehe S. chmod). Eine solche Rechtevergabe ist an sich unsinnig, aber da zu sperrende Dateien Daten enthalten und keine ausführbaren Dateien sind, brauchen sie nicht ausführbar zu sein. Der Befehl, eine Datei namens datendatei auf mandatory locking umzustellen, lautet:

```
chmod 2644 datendatei
```

Letztlich ist das mandatory locking nicht so wichtig, wie man vielleicht auf den ersten Blick glaubt. Eine Datendatei wird normalerweise mit Programmen zugegriffen, die »wissen«, wie die Daten zu behandeln sind und damit auch darauf eingestellt sind, die Sperren konkurrierender Prozesse zu beachten. Alles andere ist definitiv ein Programmierfehler. Normalerweise wird nur dann mit Fremdprogrammen auf solche Dateien zugegriffen, wenn etwas schief gelaufen ist und eine Administration notwendig ist. Und in diesen Fällen würde ein mandatory locking die Hilfe der entsprechenden Tools aussperren. Da das mandatory locking zusätzlich die Performance herabsetzt, wird es eher selten verwendet.

« [Dateieigenschaften ändern](#) | [Dateizugriffe](#) | [Link erzeugen: link, symlink](#) »

Link erzeugen: `link`, `symlink`

Links können auch aus einem Programm erstellt werden. Der Systemaufruf heißt `link()`:

```
#include <unistd.h>
int link(const char *orig, const char *neu);
```

Die Funktion `link()` erzeugt einen harten Link wie der Befehl `ln`. Auch hier gelten die Einschränkungen, die beim Befehl `ln` gelten (siehe S. [link](#)). Die Funktion gibt 0 bei Erfolg und -1 bei einem Fehler zurück.

```
#include <unistd.h>
int symlink(const char *orig, const char *neu);
```

Mit der Funktion `symlink` wird ein symbolischer Link erzeugt.

Löschen: unlink

Die Funktion zum Löschen einer Datei heißt `unlink()`, was seine Funktion auch besser beschreibt als `remove`. Tatsächlich entfernt das Löschen eine Datei nur dann, wenn der zu löschende Verzeichniseintrag der letzte Link auf die Datei ist.

```
#include <unistd.h>
int unlink(const char *pathname);
```

Die Funktion gibt 0 bei Erfolg und -1 bei einem Fehler zurück. Die Fehlerursache steht in der Variablen `errno`.

Umbenennen: rename

Die Funktion zum Umbenennen von Dateien heißt `rename()`, obwohl sie wie das Kommando `mv` wirkt. Es ist also auch möglich, damit Dateien innerhalb des gleichen Dateisystems zu verschieben.

```
#include <unistd.h>
int rename(const char *dateiname, const char *neuname);
```

Die Funktion gibt 0 bei Erfolg und -1 bei einem Fehler zurück. Die Fehlerursache steht in der Variablen `errno`.

Temporäre Dateien

Temporäre Dateien werden unter UNIX immer im Verzeichnis /tmp oder /usr/tmp abgelegt. Auf diese Verzeichnisse kann jeder frei zugreifen. Wollen Sie sicher sein, dass der Name der temporären Datei nicht auch von einem anderen Programm verwendet wird, lassen Sie ihn vom System erzeugen. Dies tut die Funktion `tmpnam()`.

```
#include <stdio.h>
char *tmpnam(char *name);
```

Ist der Parameter `name` `NULL`, so liefert die Funktion als Rückgabewert einen Zeiger auf einen internen Namen, der beim nächsten Aufruf von `tmpnam()` überschrieben wird. Ist `name` nicht `NULL`, liefert die Funktion den Namen an dieser Adresse ab.

Es gibt daneben noch die Funktion `tmpfile()`, die einen Zeiger auf `FILE` zurückgibt. Die Funktion wählt einen Namen, und eröffnet ihn sofort. Eine Besonderheit ist, dass diese Datei nach dem Schließen oder bei Programmende automatisch gelöscht wird.

```
#include <stdio.h>
FILE *tmpfile (void);
```

« [Umbenennen: rename](#) | [Dateizugriffe](#) | [Verzeichnisse](#) »

Verzeichnisse

In älteren Versionen von UNIX wurden die Verzeichnisaufrufe noch durch normale Dateioperationen realisiert, die die Datenstruktur der Verzeichniseinträge manipulierte. Diese Methode wird heute allerdings nicht mehr unterstützt.

Unterabschnitte

- [Auslesen: opendir, readdir, closedir](#)
 - [Ermitteln des Arbeitsverzeichnisses](#)
 - [Wechseln: chdir](#)
 - [Anlegen und Löschen: mkdir, rmdir](#)
-

Auslesen: opendir, readdir, closedir

Um ein Verzeichnis auszulesen, wird es zuerst mit dem Aufruf `opendir()` geöffnet, dann werden die Einträge mit `readdir()` gelesen, und schließlich wird es mit `closedir()` geschlossen. Analog zum Dateihandle gibt es ein Verzeichnishandle, das vom Variablentyp ein Zeiger auf DIR ist. Informationen über den Eintrag liefert die von `readdir()` gelieferte Struktur `dirent`.

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *pfadname);
struct dirent *readdir(DIR *dir);
int closedir(DIR *dir);
```

Die Funktion `opendir()` erhält als Parameter den Namen des Verzeichnisses. Der Rückgabewert ist ein Zeiger auf die Verzeichnisdatei, die im Sinne eines Handle behandelt wird. Ein Fehler wird dadurch angezeigt, dass dieser Zeiger den Wert 0 hat.

Die Funktion `readdir()` liest den nächsten Eintrag im Verzeichnis und erhält als Rückgabewert einen Zeiger auf eine Struktur `dirent`. Dieser Zeiger ist nur bis zum nächsten `readdir()` gültig und hat folgende Struktur:

```
struct dirent {
    long        d_ino; /* Inode Nummer */
    off_t       d_off; /* Offset zum nächsten dirent */
    unsigned short d_reclen; /* Länge dieses Eintrags */
    char        d_name[NAME_MAX+1]; /* Dateiname */
};
```

Für das Anwenderprogramm ist eigentlich nur der Name des Eintrags interessant. Wollen Sie mehr über diesen Eintrag erfahren, beispielsweise, ob es wieder ein Verzeichnis ist, so brauchen Sie dazu andere Systemaufrufe wie die Funktion `stat()` (siehe S. `stat()`).

Zuletzt wird das Verzeichnis mit `closedir()` wieder geschlossen. Ein Beispielprogramm für das Auslesen eines Verzeichnisses sieht so aus:

```
[Auslesen eines Verzeichnisses]
#include <sys/types.h>
#include <dirent.h>
```

```
int main(int argc, char **argv)
{
    DIR *dirHandle;
    struct dirent * dirEntry;
```

```
dirHandle = opendir("."); /* oeffne aktuelles verzeichnis */
if (dirHandle) {
    while (0 != (dirEntry = readdir(dirHandle))) {
        puts(dirEntry->d_name);
    }
}
```

```
        closedir(dirHandle);  
    }  
}
```

« [Verzeichnisse](#) | **Verzeichnisse** | [Ermitteln des Arbeitsverzeichnisses](#) »

Ermitteln des Arbeitsverzeichnisses

Die Funktion `getcwd()` ermittelt das aktuelle Arbeitsverzeichnis. Dazu hat das aufrufende Programm einen Puffer für den Namen zur Verfügung zu stellen, der groß genug ist. Die Größe wird als weiterer Parameter `laenge` übergeben. Reicht dieser Speicher nicht aus, gibt `getcwd()` `NULL` zurück.

```
#include <unistd.h>
char * getcwd(char *pfadname, size_t laenge);
```

In einigen Systemen ist es zulässig, eine 0 als Parameter für `namebuffer` zu übergeben. Dann alloziert `getcwd()` selbst den benötigten Speicher und gibt den Zeiger darauf zurück. Dann muss die Anwendung durch einen Aufruf von `free()` dafür sorgen, dass der Speicher wieder zurückgegeben wird.

Die alte Funktion `getwd()` wird zwar teilweise noch unterstützt, sollte aber in neuen Anwendungen nicht mehr verwendet werden.

[« Auslesen: opendir, readdir, closedir](#) | [Verzeichnisse](#) | [Wechseln: chdir](#) »

Wechseln: chdir

Mit der Funktion `chdir()` wird das aktuelle Arbeitsverzeichnis gewechselt.

```
#include <unistd.h>
int chdir(char *pfadname);
```

Bei Erfolg gibt die Funktion 0, ansonsten -1 zurück. Die Fehlernummer findet sich in der Variablen `errno`.

Anlegen und Löschen: mkdir, rmdir

Die Funktionen zum Anlegen und Löschen der Verzeichnisse heißen wie die analogen Befehle. Beim Anlegen wird analog zum `open()` bei Dateien eine Berechtigungskennung übergeben. Wie das Kommando `rmdir` kann auch die Funktion nur leere Verzeichnisse löschen.

```
#include <fcntl.h>
#include <unistd.h>
int mkdir(char *pfadname, mode_t modus);
int rmdir(char *pfadname);
```

Bei Erfolg geben die Funktionen 0, ansonsten -1 zurück. Die Fehlernummer findet sich in der Variablen `errno`.

Prozesse

Ein Prozess wird eindeutig bestimmt durch seine PID. Der Prozess hat die User-ID desjenigen, der den Prozess gestartet hat. Er kann sie mit der Funktion `getuid()` ermitteln. Gegebenenfalls läuft der Prozess unter einer anderen als der eigenen User-ID, wenn die Programmdatei das Set-User-ID Bit gesetzt hat (siehe S. `chmod`). Diese ID ermittelt die Funktion `geteuid()`.

Jeder Prozess hat eine eindeutige Prozess-ID. Diese kann er mit der Funktion `getpid()` ermitteln. Da ein Prozess immer von einem anderen Prozess gestartet wurde, hat er auch einen eindeutigen Elternprozess und auch dessen ID kann er ermitteln. Dazu gibt es die Funktion `getppid()`.

```
#include <unistd.h>
pid_t getpid(void);
pid_t getppid(void);
uid_t getuid(void);
uid_t geteuid(void);
```

Unterabschnitte

- [Multiprocessing contra Multithreading](#)
- [Vervielfältigen von Prozessen: fork](#)
 - [Geburt eines Dämons](#)
 - [Der unsterbliche Prozess](#)
- [exec und system](#)
- [Synchronisation: wait](#)
- [Prozessumgebung](#)
 - [Prozessgruppen](#)
 - [Sitzung](#)
 - [Kontrollterminal](#)
 - [Gegenseitige Abhängigkeiten](#)
- [Gemeinsamer Speicher: Shared Memory](#)
 - [Beispiel](#)
- [Synchronisation mit Semaphoren](#)
 - [Beispiel](#)
- [Message Queues](#)
 - [Beispiel](#)
- [Leichtgewichtsprozesse: Threads](#)

Multiprocessing contra Multithreading

UNIX hat ein durchaus schlankes und effizientes Prozesskonzept, das auf dem Aufruf von `fork()` basiert. Das Teilen der Prozesse mit allen Ressourcen ermöglicht es leicht, Aufgaben auf mehrere Prozesse zu verteilen. Davon machen die meisten Dämonen auch reichlich Gebrauch. Durch die Aufteilung in zwei Prozesse können sich parallele Jobs nicht so leicht gegenseitig durcheinander bringen. Dieses Konzept ist ideal für parallel laufende Serverprozesse.

Im Gegensatz zu einem Prozess arbeitet ein Thread nicht mit einem eigenen Speicherbereich, sondern teilt sich mit dem Vaterprozess alle Ressourcen. Normalerweise besteht ein Thread aus einer Funktion, die parallel zum Rest des Programms gestartet wird. Threads haben ihr Haupteinsatzgebiet im Bereich der grafischen Oberflächen. Hier ist es erforderlich, dass die grafische Darstellung betreut wird, insbesondere das Verarbeiten der Nachrichten, die über Bildneuaufbau, Mauspositionen und ähnliches berichten. Dies muss auch dann präsent sein, wenn das Programm gerade seiner eigentlichen Aufgabenstellung nachgeht und dabei vielleicht langwierige Berechnungen durchführt. Dazu ist ein Thread ideal, da er schnell ohne großen Aufwand programmiert werden kann. Da die Aufgaben der Threads fast keine Berührungspunkte miteinander haben, muss man sie auch nicht gegeneinander absichern.

Beide Konzepte haben also völlig unterschiedliche Umgebungen, in denen sie arbeiten. Die in Diskussionen manchmal anzutreffende Aussage, dass Threads so unglaublich viel performanter seien, ist also Augenwischerei. Wer einen Server durch Threads parallelisiert, wird zumindest unter UNIX soviel Verwaltungsarbeit durch die Synchronisation der Threads als Überhang bekommen, dass sich der Aufwand oft nicht lohnt. In der Konsequenz bedeutet Threading, dass jede globale Variable, jeder Dateizeiger oder sonstige Ressource, die gemeinsam genutzt wird, vor dem gegenseitigen Verändern zu schützen ist. Lediglich auf Systemen mit einem anderen Prozesskonzept, die statt dem `fork()` nur einen kompletten Neustart des Programmes kennen, wird man eventuell auf das Threading ausweichen, weil es dort sehr umständlich ist, die Daten des Vaters an den Sohn zu übermitteln.

Das Konzept mit `fork()` ist derart schnell und flexibel, dass der Thread erst in den letzten Jahren bei einigen UNIX Systemen Einzug gehalten hat. Inzwischen existiert ein POSIX-Standard für die Programmierschnittstelle der Threads. Diese wird ab Seite [thread](#) behandelt.

Unterabschnitte

- [Geburt eines Dämons](#)
- [Der unsterbliche Prozess](#)

Vervielfältigen von Prozessen: fork

Ein neuer Prozess entsteht durch den Aufruf von `fork()`. Er dupliziert den aktuell laufenden Prozess. Anschließend laufen beide Prozesse parallel. Der neue Prozess ist ein Duplikat der Arbeitsumgebung des Vaters, inklusive des Zustands der CPU, des gesamten Speicherzustands sowie aller offenen Dateien.

```
#include <unistd.h>
pid_t fork(void);
```

Beide Prozesse stehen nach dem Ausführen des `fork()` direkt hinter dem Funktionsaufruf und unterscheiden sich nicht. Nur am Rückgabewert des `fork()` erkennt der jeweilige Prozess, ob er der Vater oder der Sohn ist.

```
[Prozessteilung durch fork]
int SohnPID;
```

```
SohnPID=fork();
if (SohnPID > 0) {
    /* Der Vater ist hier aktiv */
} else if (SohnPID == 0) {
    /* Der Sohn ist hier aktiv */
} else {
    /* das war's wohl: Fehler! */
}
```

Diese Konstruktion ist ideal, um Serverprozesse zu implementieren. Sobald eine Anfrage vorliegt, teilt sich der Prozess. Beide Prozesse haben die gleichen Informationen, kennen also den Anfrager und haben die Zugriffe auf die benötigten Dateien. Der Vaterprozess kann also hier die Arbeit ohne Zeitverlust dem Sohn überlassen, die Verbindung zum Anfrager schließen und auf neue Anfragen warten.

Geburt eines Dämons

Wie schon an anderer Stelle erwähnt, ist ein Dämon ein Prozess, der im Hintergrund läuft und auf ein bestimmtes Ereignis wartet. Serverprozesse sind als Dämonen implementiert oder werden von Dämonen gestartet. Wenn ein Prozess im Hintergrund laufen soll, erzeugt er von sich selbst ein Duplikat und endet, so dass nur noch der Sohn läuft. Das Ergebnis ist, dass dem Sohn der Vater fehlt. Das macht den init-Prozess so traurig, dass er den Sohn adoptiert. Der Code ist sehr kurz:

```
if (fork()!=0) exit(0);
```

Als weiterer Vorteil gilt, dass der Prozess nicht das SIGHUP-Signal bekommt, falls er von Hand gestartet wurde und der startende Benutzer sich abmeldet. Dieses würde der Vater bekommen. Da der aber nicht mehr lebt...

Der unsterbliche Prozess

In manchen Fällen ist es wichtig, dass ein Prozess zwar aufgrund widriger Umstände auch sterben könnte, aber dann sofort wieder neu erzeugt werden soll. Auch eine solche Konstruktion können Sie mit dem `fork` leicht erzeugen.

```
[Unsterblicher Prozess]
for(;;) { /* bis zum nächsten Stromausfall */
    procid = fork();
    if (procid>0) { /* Vater */
        wait(&Zustand);
        /* wenn wir hier sind, ist der Sohn tot */
    } else { /* Sohn */
        for (;;) { /* forever and ever ... */
            /* hier arbeitet der Sohn ewig (fast)... */
        }
    }
}
```

Der Vaterprozess läuft sofort auf den Aufruf von `wait()`. Er wartet also, bis der Sohn endet. Da dieser eigentlich endlos arbeiten soll, heißt das, dass der Vaterprozess nur weiterläuft, wenn der Sohn aus welchem Grund auch immer stirbt. Der Vater wiederholt daraufhin die Schleife und kommt wieder zum Aufruf von `fork()`, erzeugt also wieder einen neuen Sohn.

Muss ein solcher Dämon doch einmal abgeschossen werden, muss natürlich der Vater vor dem Sohn getötet werden.

« [Multiprocessing contra Multithreading](#) | [Prozesse](#) | [exec und system](#) »

exec und system

Der Systemaufruf `exec()` überlädt den aktuellen Prozess mit dem Inhalt einer ausführbaren Datei und startet sie. Da der alte Speicherinhalt überschrieben wurde, gibt es kein Zurück mehr. Vor dem `exec()` wird typischerweise ein `fork()` aufgerufen. Ein Programmaufruf der Shell beispielsweise läuft so ab, dass zunächst die Shell `fork()` aufruft. Der Sohnprozess ruft nun `exec()` mit dem Programmnamen auf, der in der Shell eingegeben wurde. Der Vaterprozess dagegen ruft `wait()` auf und wartet auf das Ende des Sohnes.

Da diese Funktionalität recht häufig auftritt, gibt es einen eigenen Aufruf namens `system()`. Als Parameter erhält er den Programmnamen. Aus Sicht des Programmes wird das angegebene Programm gestartet und nach dem Ende des Programms setzt das aufrufende Programm seine Aktivität fort. Um genau zu sein, wird eine Shell (`/bin/sh`) mit dem Kommando gestartet.

```
int system(const char *programname)
```

Der Aufruf `exec()` ist eigentlich eine ganze Funktionsfamilie. Die verschiedenen Verwandten werden gebraucht, um die Aufrufparameter ordentlich weiter geben zu können.

- `[execl(char *path, char *arg, ...);]`
exec mit fester Anzahl von Argumenten. Der letzte Parameter muss NULL sein.
- `[execlp(char *file, char *arg, ...);]`
exec mit fester Anzahl von Argumenten. Der letzte Parameter muss NULL sein.
- `[execle(char *path, char *arg, ..., char *env[]);]`
exec mit fester Anzahl von Argumenten. Der vorletzte Parameter muss NULL sein. Der letzte Parameter ist ein Zeiger auf die Umgebungsvariablen.
- `[execv(char *path, *char arg[]);]`
exec mit Übernahme einer Argumentliste als Vector wie bei main. Das letzte Element des Parameterarrays arg muss NULL sein.
- `[execvp(char *file, *char arg[]);]`
exec mit Übernahme einer Argumentliste als Vector wie bei main. Das letzte Element des Parameterarrays arg muss NULL sein.

« Vervielfältigen von Prozessen: fork | Prozesse | Synchronisation: wait »

Synchronisation: wait

Der Systemaufruf `wait()` suspendiert einen Prozess solange, bis ein Kindprozess terminiert. War der Kindprozess bereits vor dem Aufruf von `wait` gestorben, hat das System in der Prozesstabelle noch die Informationen für die Anfrage des Vaters in Form eines so genannten Zombieprozesses aufbewahrt. In diesem Fall wird der Zombie aufgelöst und der `wait()` kehrt sofort zurück.

```
#include <sys/types.h>
#include <sys/wait.h>
```

```
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
```

Während `wait()` auf das Eintreffen eines beliebigen Kindprozesses wartet, wird bei `waitpid()` auf einen speziellen Prozess gewartet. Im dritten Parameter `options` können folgende Konstanten geodert übergeben werden.

[Rückkehroptionen]L|L Konstante & Bedeutung

WNOHANG & sofort zurückkehren, wenn kein Kind geendet hat

WUNTRACED & kehrt auch zurück, wenn das Kind gestoppt wurde

« [exec und system](#) | **Prozesse** | [Prozessumgebung](#) »

Unterabschnitte

- [Prozessgruppen](#)
- [Sitzung](#)
- [Kontrollterminal](#)
- [Gegenseitige Abhängigkeiten](#)

Prozessumgebung

Die Informationen über Prozessgruppe, Sitzung und Kontrollterminal von Prozessen können Sie sich mit `ps` anschauen, wenn Sie die Optionen `-jx` bzw. unter System V `-jl` angeben. Dabei stehen die wichtigen Informationen unter folgenden Überschriften.

[ps-Überschriften]L|L Kürzel & Bedeutung
PGID & Prozessgruppen ID
SID & Sitzung ID
TTY & Kontrollierendes Terminal
TPGID & Gruppen ID des kontrollierenden Terminal

Prozessgruppen

Jeder Prozess gehört zu einer Prozessgruppe. Die Prozessgruppen-ID ist die Prozess-ID des Prozessgruppenleiters. Eine Prozessgruppe kann beispielsweise durch eine Pipe verbunden sein. Anders ausgedrückt ist eine Prozessgruppe eine Gruppe von Prozessen, die voneinander abhängig sind und in dieser Abhängigkeit zusammengehören. Durch die folgende Kommandosequenz würde man die in der grafischen Übersicht gezeigten Prozessgruppen erzeugen:

```
gaston> processA | processB &  
[1] 2354  
gaston> processD | processG | prozessK &  
[2] 2356  
gaston> processM | processN
```

Die Prozessgruppe kann mit dem Aufruf `getpgrp()` ermittelt werden. Mit dem Aufruf `getpgid()` kann die Prozessgruppe eines anderen Prozesses ermittelt werden, dessen ID als Parameter übergeben wird.

Mit dem Aufruf `setpgid()` kann einem Prozess eine neue Prozessgruppe zugewiesen werden. Ein Prozess kann `setpgid()` nur für sich selbst oder einen seiner Kindprozesse aufrufen, solange das Kind noch nicht `exec` gerufen hat.

```
#include <unistd.h>  
int setpgid(pid_t pid, pid_t pgid);  
pid_t getpgid(pid_t pid);  
int setpgrp(void);  
pid_t getpgrp(void);
```

Sitzung

Einer Sitzung (engl. session) können ein oder mehrere Prozessgruppen gehören. Mit `setsid()` wird eine neue Sitzung eröffnet und damit auch eine neue Prozessgruppe.

```
#include <unistd.h>
pid_t setsid(void);
```

Eine Sitzung besteht aus einer Vordergrundprozessgruppe und einer beliebigen Zahl von Hintergrundprozessgruppen.

Kontrollterminal

Das Kontrollterminal ist das Terminal oder das Pseudoterminal, von dem aus eine Sitzung ursprünglich gestartet wurde. Sofern ein Prozess noch ein Kontrollterminal hat, ist dieses für ihn jeweils über `/dev/tty` zu erreichen.

Gegenseitige Abhängigkeiten

Versucht ein Hintergrundprozess in einer Umgebung mit Jobkontrolle vom Terminal zu lesen, wird er suspendiert, bis der Prozess durch die Jobkontrolle explizit in den Vordergrund geholt wird. Existiert keine Jobkontrolle, erhält ein Hintergrundprozess beim Versuch, vom Terminal zu lesen, `/dev/null` zugewiesen. Da dies immer EOF (End Of File) liefert, wird die Eingabe sofort abgeschlossen.

Die Ausgabe von Hintergrundprozessen ist grundsätzlich zugelassen, kann aber durch das Kommando `stty tostop` unterbunden werden. Um die Ausgabe wieder zuzulassen, geben Sie das Kommando `stty -tostop`.

« [Synchronisation: wait](#) | [Prozesse](#) | [Gemeinsamer Speicher: Shared Memory](#) »

Unterabschnitte

- [Beispiel](#)

Gemeinsamer Speicher: Shared Memory

Normalerweise sind die Speicherbereiche zweier Prozesse streng getrennt. Der gemeinsame Speicher ermöglicht die Arbeit zweier Prozesse an den gleichen Daten. Neben den hier beschriebenen Prinzipien, sich den Speicher zu teilen, benötigen Sie normalerweise eine Form der Synchronisation, mit der die Prozesse sich darüber austauschen, wann wer auf den Speicher zugreifen darf.

Die Funktion `shmget()` legt den gemeinsamen Speicher an, bzw. eröffnet ihn.

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, int size, int shmflg);
```

Der Parameter `key` ist entweder eine Schlüsselzahl oder `IPC_PRIVATE`.

Der Parameter `shmflg` kann die Konstanten `IPC_CREAT` und `IPC_EXCL` und neun Berechtigungsbits für den Eigner, die Gruppe und der Welt, aufnehmen. Die Berechtigungen sind von dem Befehl `chmod` bekannt. Man kombiniert die Werte, indem man sie mit dem senkrechten Strich ordert.

Der Rückgabewert ist -1 im Fehlerfall oder die Shared Memory ID, die für die nächsten Aufrufe benötigt wird.

Die Funktion `shmat()` (shared memory attach) bindet den Speicher ein. Mit `shmdt()` (shared memory detach) wird die Speicherbindung wieder aufgehoben.

```
# include <sys/types.h>
# include <sys/shm.h>
void *shmat(int shmid, const void *shmaddr, int shmflg);
int shmdt(const void *shmaddr);
```

Der Parameter `shmid` ist die von `shmget()` ermittelte ID. An den Parameter `shmaddr` kann eine 0 übergeben werden, dann sucht sich das System eine passende Stelle. Die `shmflg` ist 0 oder `SHM_RDONLY`, wenn der Speicher nur lesend zugegriffen werden soll. Nach `shmat()` steht der Speicher zur Verfügung und kann wie ein normaler Speicherbereich zugegriffen werden.

Die Fehlermeldung von `shmat()` ist (leider) -1. Da der Rückgabewert ein Zeiger ist, der nicht mit einer natürlichen Zahl verglichen werden darf, ergeben sich immer Abfragen wie:

```
myPtr = shmat(shID, 0, 0);
if (myPtr==(char *)-1)
```

Mit der Funktion `shmctl()` werden bestimmte Eigenschaften des gemeinsamen Speichers verwaltet.

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmctl(int shmid, int kommando, struct shmid_ds *buf);
```

An den Parameter `kommando` können folgende Konstanten übergeben werden:

[Shared Memory Kontrollkommandos] L | L Konstante & Bedeutung
IPC_STAT & Die Informationen über den Speicher einlesen
IPC_SET & Ändere die Benutzerrechte in mode
IPC_RMID & Markiere das Segment als zerstört

Mit dem Kommandozeilenbefehl `ipcs` bekommen Sie einen Überblick über die angeforderten Shared Memory Bereiche.

Beispiel

Das Programm `one` erzeugt einen Shared Memory von 30 Byte und schreibt dort ASCII-Zeichen beginnend mit A hinein.

```
[Shared Memory one.c]
#include <sys/ipc.h>
#include <sys/shm.h>

#define MAXMYMEM 30

int main(int argc, char **argv)
{
    int shID;
    char *myPtr;
    int i;

    /* Shared Memory erzeugen */
    shID = shmget(2404, MAXMYMEM, IPC_CREAT | 0666);
    if (shID >= 0) {
        /* nun holen wir den Speicher */
        myPtr = shmat(shID, 0, 0);
        if (myPtr==(char *)-1) {
            perror("shmat");
        } else {
            /* Speicher ist zugreifbar: füllen! */
            for (i=0; i<MAXMYMEM; i++) {
                myPtr[i] = 'A'+i;
            }
            getchar(); /* warte mal auf eine Taste */
            /* gebe den Speicher auf */
            shmdt(myPtr);
        }
    } else { /* shmget lief schief */
        perror("shmget");
    }
}
```

Das Programm two unterscheidet sich wenig von one. Da Programm one den Speicher erzeugt, braucht two das nicht zu tun. two wird einfach den Inhalt des Speichers auslesen und auf dem Bildschirm ausgeben und damit demonstrieren, dass es sich um denselben Speicher handelt.

```
[Shared Memory two.c]
#include <sys/ipc.h>
#include <sys/shm.h>
#define MAXMYMEM 30

int main(int argc, char **argv)
{
    int shID;
    char *myPtr;
    int i;

    /* Existierenden Shared Memory zugreifen */
    shID = shmget(2404, MAXMYMEM, 0666);
    if (shID >= 0) {
        myPtr = shmat(shID, 0, 0);
        if (myPtr==(char *)-1) {
            perror("shmat");
        } else {
            for (i=0; i<MAXMYMEM; i++) {
                putchar(myPtr[i]);
            }
            puts("\n");
            shmdt(myPtr);
        }
    } else { /* shmget lief schief */
        perror("shmget");
    }
}
```

Die Programme brauchen keineswegs parallel zu laufen. Man kann one auch durchlaufen lassen und sieht dann mit dem Befehl `ipcs`, dass der Shared Memory noch da ist. Um den Speicher zu entsorgen, muss er zerstört werden. Dazu dient das kleine Programm `destroy`:

```
[Shared Memory destroy.c]
#include <sys/ipc.h>
#include <sys/shm.h>
#define MAXMYMEM 30

int main(int argc, char **argv)
{
    int shID;
    char *myPtr;
    int i;

    /* Shared Memory erzeugen */
    shID = shmget(2404, MAXMYMEM, 0666);
    if (shID >= 0) {
        /* zerstöre den Shared Memory */
        shmctl(shID, IPC_RMID, 0);
    } else { /* shmctl lief schief */
        perror("shmctl");
    }
}
```

Ein interessantes Experiment zeigt das parallele Starten von `one` und `destroy`. Sie starten `one` auf einem Terminal. Dieses legt den Shared Memory an und bindet ihn ein. Dann wartet das Programm. Nun starten Sie `destroy` auf der anderen Konsole. Der Aufruf zum Zerstören des Speichers hat stattgefunden. Aber ein Blick auf die Ausgabe von `ipcs` zeigt, dass der Speicher noch da ist. Erst wenn `one` mit einem Tastendruck weiterläuft und seinen Speicher wieder mit `shmdt()` freigegeben hat, zeigt auch `ipcs` das Verschwinden des Shared Memory.

Bei dem Beispiel wird die Synchronisation der Programme durch den zeitlich unterschiedlichen Start erreicht. Bei einem parallelen Start von `one` und `two` wäre es aber denkbar, dass `one` noch gar nicht damit fertig wäre, die Daten in den Speicher zu schreiben, während `two` bereits mit dem Auslesen beginnt. Um dies zu verhindern, wird ein weiteres Konzept benötigt, das gewährleisten kann, dass der kritische Bereich nur von einem Programm gleichzeitig bearbeitet wird.

Der Shared Memory bleibt solange erhalten, bis ein Programm ihn explizit entfernt (s. o.), bis zum nächsten Shutdown oder bis er mit dem Befehl `ipcrm` explizit gelöscht wird.

```
gaston> ipcs
```

```
--- Shared Memory Segments ----
```

key	shmid	owner	perms	Bytes	nattch	Status
0x00000964	425987	arnold	666	30	0	

```
--- Semaphore Arrays ----
```

key	semid	owner	perms	nsems	Status
-----	-------	-------	-------	-------	--------

```
--- Message Queues ----
```

key	msqid	owner	perms	used-bytes	messages
-----	-------	-------	-------	------------	----------

```
gaston> ipcrm shm 425987
```

```
resource(s) deleted
```

```
gaston>
```

« [Prozessumgebung](#) | [Prozesse](#) | [Synchronisation mit Semaphoren](#) »

Unterabschnitte

- [Beispiel](#)

Synchronisation mit Semaphoren

Semaphoren gehen auf den Informatiker E. W. Dijkstra zurück. Sie dienen als Schutz gegen gleichzeitiges Operieren in einem kritischen Bereich. Wenn beispielsweise der gemeinsame Speicher (Shared Memory) von zwei oder mehr Prozessen gleichzeitig genutzt wird, muss verhindert werden, dass sie gleichzeitig schreiben oder dass ein Prozess liest, während ein anderer Prozess schreibt.

Das Erzeugen einer Semaphore erfolgt analog zum Erzeugen des Shared Memory. Selbst die Konstante `IPC_CREAT` ist identisch.

```
#include <sys/ipc.h>
#include <sys/sem.h>
int semget(key_t key, int nsems, int semflg);
```

Der Parameter `key` ist wie bei Shared Memory eine Zahl, die von den Programmen vereinbart wird, die über die Semaphore kommunizieren wollen. Im zweiten Parameter wird angegeben, wieviele Semaphoren erzeugt werden sollen.

Der letzte Parameter enthält die Berechtigung, wie man sie von `chmod` kennt (siehe S. `chmod`). Mit dieser wird die Konstante `IPC_CREAT` geodert, damit die Semaphore angelegt wird, falls sie noch nicht existierte. Soll der Aufruf scheitern, wenn es bereits eine solche Semaphore gibt, verknüpfen Sie auch die Konstante `IPC_EXCL` mit Oder.

Der Rückgabewert ist die Semaphoren-ID, die für die Identifikation benötigt wird. Scheiterte der Aufruf, gibt er -1 zurück.

Die Semaphoren werden mit dem Aufruf von `semop()` bearbeitet.

```
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, unsigned anzahl);
```

Als Parameter `semid` wird der Rückgabewert von `semget()` verwendet. Der zweite Parameter nimmt die Adresse eines Array von Strukturen `sembuf` auf, die je eine Semaphorenoperation beschreiben. Wieviele Elemente in dem Array stehen, gibt der dritte Parameter `anzahl`.

Die Struktur `sembuf` beinhaltet die folgenden Felder:

```
struct sembuf {
    short sem_num; /* semaphore number: 0 = first */
    short sem_op;  /* semaphore operation */
```

```
    short sem_flg; /* operation flags */  
};
```

sem_num gibt an, welche Semaphore des Semaphorensets gemeint ist. Die Zählung beginnt bei 0. sem_flg kann die Optionen IPC_NOWAIT und SEM_UNDO annehmen. SEM_UNDO bewirkt, dass die Operation bei Ende des Prozesses zurückgenommen wird.

Der Systemaufruf semop() gewährleistet, dass die Operationen nur durchgeführt werden, wenn alle Operationen gelingen.

Die Variable sem_op der Struktur sembuf bestimmt die Operation.

- [semop>0]
Der Wert der Semaphore wird um den Wert von semop erhöht. Diese Operation blockiert nie.
- [semop==0]
Wenn der Wert der Semaphore (semval) Null ist, läuft die Operation durch. Ansonsten blockiert der Prozess, bis die Semaphore Null wird.
- [semop<0]
Der Wert der Semaphore wird um den Wert von semop verringert, sofern die Semaphore durch diese Operation nicht negativ wird. Ist semop größer als der Wert der Semaphore, schläft der Prozess bis der Wert hoch genug ist.

```
#include <sys/ipc.h>  
#include <sys/sem.h>  
int semctl(int semid, int semnum, int kommando,  
           union semun arg);
```

Der Parameter kommando gibt an, welche Operation ausgeführt wird. Das wichtigste Kommando ist IPC_RMID. Es zerstört sofort alle Semaphoren dieses Semaphorensets. Die auf die Semaphoren wartenden Prozesse werden geweckt und erhalten einen Fehler als Rückgabewert ihres Funktionsaufrufs.

Beispiel

Das Programm sem erzeugt eine Semaphore, sofern nicht schon eine existiert, und wartet auf die Returntaste. Anschließend setzt sie die Semaphore. Das führt zum Blockieren oder das Programm betritt den kritischen Bereich. Das wird am Bildschirm angezeigt. Nach erneutem Drücken der Returntaste verlässt das Programm wieder den kritischen Bereich und gibt ihn für andere frei. Zum Testen können Sie das Programm auf mehreren Terminals oder Fenstern starten und sie nacheinander in den kritischen Bereich gehen lassen.

```
[Semaphoren sem.c]  
#include <sys/ipc.h>  
#include <sys/sem.h>  
  
int main(int argc, char **argv)  
{  
    int semID;  
    struct sembuf sema;  
  
    /* Semaphore erzeugen */  
    semID = semget(2404, 1, IPC_CREAT | 0666);  
    if (semID >= 0) {  
        puts("Semaphore erzeugt. Vor Anfrage");
```

```

getchar();
/* Bereite die Semaphore vor und starte */
sema.sem_num = 0;
sema.sem_flg = SEM_UNDO;
sema.sem_op = -1;
if (-1==semop(semID, &sema, 1)) {
    /* Fehler */
    perror("semop");
}
puts("bin im kritischen Bereich");
getchar();
sema.sem_op = 1;
if (-1==semop(semID, &sema, 1)) {
    /* Fehler */
    perror("semop");
}
puts("und nun wieder draußen");
} else {
    perror("semget");
}
}

```

Auch die Semaphore bleibt nach Verlassen des Programms bestehen und muss explizit gelöscht werden. Dazu reicht das folgende Programmbeispiel aus:

```

[Semaphoren destroy.c]
#include <sys/ipc.h>
#include <sys/shm.h>

int main(int argc, char **argv)
{
    int semID;
    char *myPtr;
    int i;

    semID = semget(2404, 1, 0666);
    if (semID >= 0) {
        /* zerstöre die Semaphore */
        semctl(semID, 1, IPC_RMID, 0);
    } else { /* semctl lief schief */
        perror("semget");
    }
}

```

Die Semaphore bleibt solange erhalten, bis ein Programm in explizit entfernt (s. o.), bis zum nächsten Shutdown oder bis er mit dem Befehl `ipcrm sem` explizit gelöscht wird (siehe S. `ipcrm`).

Unterabschnitte

- [Beispiel](#)

Message Queues

Message Queues dienen zum Senden und Empfangen von Nachrichten. Eine solche Nachrichtenschlange kann mit Nachrichten verschiedenen Typs umgehen. Der Typ wird durch die Anwendung bestimmt und ist einfach eine Zahl. Ein Prozess kann Nachrichten an die Warteschlange senden. Beim Erreichen der Kapazität der Schlange kann der Prozess per Parameter bestimmen, ob er blockieren will bis die Nachricht abzuliefern ist oder lieber mit einem Fehler zurückkehren möchte. Auf der anderen Seite kann ein Prozess eine Nachricht bestimmten Typs anfordern. Auch hier steht es dem Programmierer frei, ob er möchte, dass der Prozess wartet, bis er eine passende Nachricht bekommt oder ob er mit einer Fehlermeldung sofort zurückkehren soll.

Die Funktion `msgget()` legt eine Message Queue an.

```
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget(key_t key, int msgflg);
```

Der Parameter `key` ist entweder eine Schlüsselzahl oder `IPC_PRIVATE`. Der Parameter `msgflg` kombiniert die Konstanten `IPC_CREAT` und `IPC_EXCL` und neun Berechtigungsbits für den Eigner, die Gruppe und der Welt, wie sie von `chmod` verwendet werden, indem sie mit dem senkrechten Strich geodert werden.

Der Rückgabewert ist -1 im Fehlerfall oder die Message Queue ID, die für die nächsten Aufrufe benötigt wird.

Die Funktionen `msgsnd()` und `msgrcv()` verwenden eine Struktur `msgbuf` für ihre Nachrichten, die den Typ und den Puffer enthält.

```
struct msgbuf {
    long mtype;      /* von der Anwendung definierbar > 0 */
    char mtext[1];   /* Nachrichtendaten beginnen hier */
};
```

Es kann als Typ eine beliebige Zahl größer Null verwendet werden, die allein von der Applikation festgelegt werden. Auf diese Weise können Sie leicht verschiedene Arten von Daten austauschen und sie über den Nachrichtentyp trennen. Für die eigenen Nachrichten werden Sie im `mtext` vermutlich mehr als ein Zeichen versenden wollen. Dazu definieren Sie sich eine eigene Struktur mit entsprechend größerem Datenpuffer. Die Größe wird beiden Funktionen als Parameter übergeben.

Mit der Funktion `msgsnd()` werden Nachrichten versandt.

```
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd(int msqid, struct msgbuf *msgp, size_t msgsz,
           int msgflg);
```

Der erste Parameter ist der Rückgabewert der Funktion `msgget()`. Es folgt die Adresse der Datenstruktur mit dem Nachrichtentyp und den Daten. Der Parameter `msgsz` ist so groß wie das Array `mtext` in der Datenstruktur für die Nachricht. `msgflg` kann mit der Optionen `IPC_NOWAIT` besetzt werden. Dann wird die Funktion bei einer übervollen Message Queue nicht blockieren und warten, bis wieder Platz ist, sondern mit einem Fehler zurückkehren.

Mit der Funktion `msgrcv()` werden werden Nachrichten empfangen.

```
#include <sys/ipc.h>
#include <sys/msg.h>
int msgrcv(int msqid, struct msgbuf *msgp, size_t msgsz,
           long msgtyp, int msgflg);
```

Der erste Parameter ist der Rückgabewert der Funktion `msgget()`. Es folgt die Adresse der Datenstruktur, in der sich nach erfolgreichem Empfang der Nachrichtentyp und die Daten wiederfinden. Der Parameter `msgsz` ist so groß wie das Array `mtext` in der Datenstruktur für die Nachricht. Im Parameter `msgtyp` kann festgelegt werden, auf welchen Nachrichtentyp `msgrcv()` warten soll. Alle anderen Typen werden von `msgrcv()` ignoriert. Wird als Parameter hier 0 angegeben, nimmt `msgrcv()` jeden Typ entgegen. `msgflg` kann mit der Optionen `IPC_NOWAIT` besetzt werden. Dann wird die Funktion nicht blockieren und warten, bis eine Nachricht vorliegt, sondern bei leerer Message Queue mit einem Fehler zurückkehren.

Mit der Funktion `msgctl()` werden bestimmte Eigenschaften der Nachrichten verwaltet.

```
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl(int msqid, int kommando, struct msqid_ds *buf);
```

Als Paramter `kommando` können folgende Konstanten übergeben werden:

[Message Kontrollkommandos]

L	Konstante	Bedeutung
IPC_STAT	&	Die Informationen über die Message Queue einlesen
IPC_SET	&	Ändere die Benutzerrechte in mode
IPC_RMID	&	Zerstört die Message Queue und weckt alle darauf wartenden Prozesse

Mit dem Kommando `ipcs` bekommen Sie einen Überblick über die angeforderten Message Queues.

Beispiel

Das Programm `rcvmsg.c` wartet auf eine Nachricht in einer Message Queue. Die Nummer des Typs wird als erster Parameter beim Aufruf übergeben. Wird nichts übergeben wartet das Programm auf eine beliebige Nachricht.

```
[Empfange Nachricht rcvmsg.c]
#include <sys/ipc.h>
#include <sys/msg.h>
```

```

#define MSGSIZE 20

int main(int argc, char **argv)
{
    int msgID;
    struct myMsg {
        long mtype;
        char mtext[MSGSIZE];
    } dataMsg;
    long msgTyp = 0;

    /* hole die Messagetyppnummer aus dem ersten Parameter */
    if (argc>1) {
        msgTyp = atol(argv[1]);
    }

    /* Messagequeue oeffnen bzw. erzeugen */
    msgID = msgget(2404, IPC_CREAT | 0666);
    if (msgID >= 0) {
        printf("Warte auf Message Type %ld\n", msgTyp);
        if (-1==msgrcv(msgID, &dataMsg, MSGSIZE, msgTyp, 0)) {
            perror("msgrcv"); /* Fehler */
        } else {
            /* wir sind durchgelaufen */
            printf("Daten empfangen: %sn", dataMsg.mtext);
        }
    } else {
        perror("msgget");
    }
}

```

Das Programm sndmsg.c sendet Nachrichten. Der Nachrichtentyp wird wie bei rcvmsg.c als erster Parameter übergeben. Als zweiter Parameter kann ein String übergeben werden, der dann als Daten in der Messages Queue abgestellt wird und den rcvmsg dann empfängt.

```

[Sende Nachrichten sndmsg.c]
#include <sys/ipc.h>
#include <sys/msg.h>

#define MSGSIZE 20

int main(int argc, char **argv)
{
    int msgID;
    struct myMsg {
        long mtype;
        char mtext[MSGSIZE];
    } dataMsg;
    long msgTyp = 0;

    /* hole die Messagetyppnummer aus dem ersten Parameter */
    if (argc>1) {
        dataMsg.mtype = atol(argv[1]);
    }
    if (argc>2) {
        strncpy(dataMsg.mtext, argv[2], MSGSIZE);
    }
}

```

```

} else {
    *dataMsg.mtext = 0;
}
/* Messagequeue oeffnen bzw. erzeugen */
msgID = msgget(2404, IPC_CREAT | 0666);
if (msgID >= 0) {
    printf("Sende Messagetyp %ldn", dataMsg.mtype);
    if (-1==msgsnd(msgID, &dataMsg, MSGSIZE, 0)) {
        perror("msgsnd"); /* Fehler */
    } else {
        /* wir sind durchgelaufen */
        printf("Daten gesendet: %sn", dataMsg.mtext);
    }
} else {
    perror("msgget");
}
}

```

Mit Hilfe der beiden Programme läßt sich das Verhalten der Message Queue leicht testen. Eine Anpassung an eigene Befürfnisse dürfte eine leichte Übung sein.

Die Message Queue bleibt solange erhalten, bis ein Programm sie explizit per `msgctl()` mit dem Kommando `IPC_RMID` entfernt, bis zum nächsten Shutdown oder bis sie mit dem Befehl `ipcrm msg` gelöscht wird (siehe S. `ipcrm`).

« [Synchronisation mit Semaphoren](#) | [Prozesse](#) | [Leichtgewichtsprozesse: Threads](#) »

Leichtgewichtsprozesse: Threads

Relativ neu ist die Möglichkeit, unter UNIX mit Threads zu arbeiten. Es gab zunächst unterschiedliche Herstellerstandards für die Programmierschnittstelle von Sun, Unisys und SGI. Linux definierte einfach Prozesse, die sich Ressourcen teilten und erzeugte diese Quasi-Threads mit dem Aufruf `clone()`. Inzwischen gibt es einen Standard nach POSIX 1003.1c, der sich mit der Zeit sicher durchsetzen wird und dafür sorgt, dass Threads auf allen UNIX-Systemen gleich programmiert werden.

Threads werden dort eingesetzt, wo parallele Vorgänge benötigt werden, bei denen möglichst wenig Synchronisation erforderlich ist oder in Situationen in denen die parallelen Vorgänge sich viele Ressourcen teilen müssen. Im Gegensatz zu Prozessen teilen sich Threads alle Ressourcen bis auf den Programmzeiger. Das bedeutet, dass die Änderung einer globalen Variablen alle Threads betrifft. Wird der Dateizeiger mit `lseek()` verändert, ist er für alle Threads verändert. Schließt ein Thread eine Datei, erhält ein paralleler Thread einen Fehler, wenn er auf diese Datei zugreift. Sie müssen auch sicherstellen, dass die Bibliotheken, die Sie benutzen, nicht Probleme mit paralleler Abarbeitung haben. Wer mit Threads arbeiten will, muss also nicht nur die Aufrufe kennen, sondern auch sehr sorgfältig arbeiten.

Der Prozess selbst ist immer auch ein Thread. Der Start eines Threads erzeugt also bereits einen zweiten Thread und damit Parallelität. Der Code für diesen zweiten Thread wird als gewöhnliche Funktion geschrieben. Die Funktion wird als Thread gestartet und sobald sie endet, endet auch der Thread. Die Funktion hat einen Zeiger als Parameter. Mit diesem Zeiger können Sie dem Thread beim Start Daten übergeben. Die Thread-Funktion liefert einen Zeiger zurück, mit dem er Daten wieder zurückgeben kann.

Der Aufruf zum Erzeugen eines Threads heißt `pthread_create()`. Dabei wird ein neuer Thread erzeugt und sofort gestartet. Da der Prozess selbst auch einen Thread darstellt, laufen dann also zwei Threads. Der wichtigste Parameter ist der Name der Funktion, die als Thread laufen soll.

```
#include <pthread.h>
int  pthread_create(pthread_t *TID, pthread_attr_t *Attribut,
                    void * (*Funktion)(void *), void *Argument);
```

Der Parameter TID kann man als Thread-ID bezeichnen. Dazu legen Sie eine Variable vom Typ `pthread_t` an und übergeben deren Adresse an die Funktion. Der zweite Parameter Attribut kann verwendet werden, um den Wechsel zwischen den Threads zu verändern. Das ist beispielsweise wichtig, wenn Sie den Wechsel in Echtzeit brauchen. Im Normalfall wird hier NULL übergeben. Für nähere Informationen zu den Attributen finden Sie Informationen in der Manpage von `pthread_attr_t`. Der dritte Parameter ist die Funktion, die den Code für den Thread enthält. Der letzte Parameter zeigt auf die Daten, die dem Thread als Parameter übergeben werden sollen. Werden keine Übergaben gebraucht, kann der Parameter mit NULL besetzt werden.

Der Rückgabewert ist 0, wenn alles in Ordnung ist.

Ein Thread endet, wenn seine Funktion endet. Damit verhält er sich ähnlich wie die Funktion `main` bei einem Prozess. Der Thread kann sich auch selbst beenden. Dazu gibt es analog zur Funktion `exit()` die Funktion `pthread_exit()`.

```
#include <pthread.h>
void pthread_exit(void *Rueckgabe);
```

Im Gegensatz zu `exit()`, der nur eine Fehlernummer zurückgibt, kann `pthread_exit()` einen Zeiger auf Daten zurückgeben. Dies entspricht dem Rückgabewert der Funktion, wenn sie regulär endet.

Diesen Zeiger bekommt derjenige Thread, der mit Hilfe der Funktion `pthread_join()` den erzeugten Thread wieder einsammelt.

```
#include <pthread.h>
int pthread_join(pthread_t TID, void **Rueckgabe);
```

Die Funktion `pthread_join()` hat zwei Aufgaben. Solange der Thread TID noch läuft wird der aufrufende Thread blockiert. Läuft der Thread nicht mehr, wird der Thread TID mit dem aktuellen Thread wieder zusammengeführt. Die letzten Reste des Threads verschwinden also. Durch den Parameter `Rueckgabe` ist auch ein Zugriff auf den Rückgabewert des Threads möglich. Dazu muss eine Zeigervariable definiert werden und deren Adresse als zweiter Parameter übergeben werden. Nach Ende des Threads kann über diese Zeigervariable auf die Rückgabedaten referenziert werden. Interessieren Sie sich nicht für den Rückgabewert, setzen Sie den zweiten Parameter auf `NULL`.

Das folgende Beispiel startet drei Threads. Die Funktion `threadPlus` enthält den Code für jeden Thread. Der Thread bekommt beim Aufruf einen Namen als Parameter. Dann läuft er in eine Schleife, die endet, wenn die Variable `stopp` auf einen Wert ungleich 0 gesetzt wird. Damit wird die Variable zur Endebedingung für alle Threads.

Innerhalb des Threads wird die globale Variable `wert` erhöht wenn der Name des Threads mit dem Buchstaben A beginnt, ansonsten wird der Inhalt der Variablen erniedrigt.

```
[Multithreading]
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

int stopp = 0; /* Synchronisation */
int wert = 0; /* Globale variable */

void *threadPlus(void *Arg)
{
    /* uebergeben wird der Name */
    char *Name = (char *)Arg;
    int diff; /* Differenz fuer wertberechnung */

    if (Name[0]=='A') {
        diff=1;
    } else {
        diff=-1;
    }
}
```

```
for (;;) {
    if (stopp) break; /* Thread endet durch Setzen von stopp */
    printf("%s: %dn", Name, wert);
    /* Namen bestimmt, was der Thread tut */
    wert += diff;
    sleep(1);
}
return NULL;
}
```

```
int main(void)
{
```

```
pthread_t Add1, Add2, Minus1;
```

```
/* Erzeuge und starte drei Threads */
pthread_create(&Add1, NULL, threadPlus, "Add1");
pthread_create(&Add2, NULL, threadPlus, "Add2");
pthread_create(&Minus1, NULL, threadPlus, "Minus1");
sleep(20); /* lass die Threads arbeiten */
stopp = 1; /* beende das Treiben */
/* warte auf das Ende der Threads und loese sie auf */
pthread_join(Add1, NULL );
pthread_join(Add2, NULL );
pthread_join(Minus1, NULL );
return 0;
}
```

Die drei Threads bekommen jede ihre eigene Thread-ID. Sie werden im Hauptprogramm gestartet. Der Hauptthread legt sich dann für 20 Sekunden schlafen und lässt die Threads addieren und subtrahieren. Durch das Setzen der Variablen stopp löst er das Ende der Threads aus. Danach führt er alle drei Threads wieder zusammen, bzw. wartet deren Ende ab.

Beim Ablauf des Programms erscheint meist Add1, dann Add2 und Minus1 in der normalen Reihenfolge. Man kann aber auch sehen, dass die Reihenfolge wechselt. An der Variablen Wert können Sie erkennen, dass die Veränderung des Wertes davon abhängt, wer gerade an die Reihe kommt.

Zum Linken des Programmes muss die Option -lpthread an den Compileraufruf angehängt werden.

« [Message Queues](#) | [Prozesse](#) | [Signale](#) »

Signale

Ein Prozess kann von außen mittels kill-Befehl Signale empfangen. Normalerweise endet der Prozess, wenn er ein solches Signal bekommt. Man kann Programme auch so schreiben, dass sie durch einen Signalbehandler auf die Signale reagieren.

Der Funktion `signal()` meldet dem Betriebssystem, dass eine Funktion des Programmes bestimmte Signale bearbeiten will.

```
#include <signal.h>
signal(int signalnr, void (*signalfunktion)(int));
```

Ein Signal, auf das ein Programm immer reagieren sollte ist SIGTERM. Er wird z. B. beim Herunterfahren des Systems an jeden Prozess gesandt. Die typische Reaktion sollte es sein, die Aktivitäten schnellstmöglich einzustellen und einen konsistenten Zustand der Daten zu gewährleisten. Dafür hat der Prozess im Falle des Herunterfahrens höchstens 5 Sekunden Zeit. Dann wird er durch einen SIGKILL endgültig zerlegt.

Besonders bei Hintergrundprozessen ist es inzwischen üblich, das SIGHUP"-Signal zu verwenden, um den Prozess dazu zu bewegen, Konfigurationsdateien neu zu lesen. So ist es möglich, Einstellungen zu ändern, ohne den Betrieb zu unterbrechen.

Tabelle benennt alle Signalkonstanten, die der erste Parameter annehmen kann.

[Signale]	Name & Bedeutung
SIGHUP	& (1) Hangup: Terminalabschaltung oder Konfiguration neu einlesen
SIGINT	& Unterbrechung durch ctrl-C oder Delete
SIGQUIT	& Unterbrechung durch ctrl-
SIGILL	& Illegale Anweisung
SIGTRACE	& Im Debugmodus
SIGIOT	& I/O Trap
SIGKILL	& (9) nicht abfangbarer Tötungsaufruf
SIGBUS	& Busfehler
SIGSEGV	& Segmentation Violation
SIGPIPE	& Schreiben auf ein nicht zum Lesen geöffnete Pipe
SIGALRM	& Aufgesetzter Alarm
SIGTERM	& (15) Terminierung
SIGUSR1	& (16) Benutzerdefiniertes Signal zur freien Verfügung
SIGUSR2	& (17) Benutzerdefiniertes Signal zur freien Verfügung
SIGCLD	& Tod eines Sohnprozesses
SIGPWR	& (19) Spannungsproblem

Die eingeklammerten Zahlen sind die Nummern der Signale, soweit sie über die Systeme hinweg einheitlich sind.

Das folgende Programm fängt das Signal SIGHUP und gibt bei jedem kill eine Meldung auf dem

Bildschirm aus.

```
[SIGHUP-Signal fangen]
#include <signal.h>

void SigHandler(int Nr)
{
    puts("Signal gefangen");
}

main()
{
    signal(SIGHUP, SigHandler);
    for (;;) ;
}
```

Unterabschnitte

- [Signale senden: kill](#)
 - [Auf Signale warten: pause](#)
 - [Timeout setzen: alarm](#)
 - [Zombies vereiteln](#)
-

« [Leichtgewichtsprozesse: Threads](#) | [UNIX-Systemaufrufe](#) | [Signale senden: kill](#) »

Signale senden: kill

Auch aus einem Programm heraus können Signale mit der Funktion `kill()` versendet werden.

```
#include <sys/types.h>
#include <signal.h>
int kill(int pid, int signal);
```

Der Parameter `pid` gibt die Prozessnummer des Empfängers an. Der Parameter `signal` bezeichnet das zu sendende Signal. Im Erfolgsfall gibt die Funktion 0, ansonsten -1 zurück.

Auf Signale warten: pause

Der Aufruf von `pause()` blockiert den Prozess und wartet auf ein beliebiges Signal.

```
#include <unistd.h>
int pause(void);
```

Dieser Aufruf liefert immer -1 als Rückgabewert.

Timeout setzen: alarm

Mit dem Aufruf `alarm()` wird ein Alarm aufgesetzt. Sobald die als Parameter übergebenen Sekunden vergangen sind, wird dem Prozess das Signal `SIGALRM` zugesandt.

Diese Funktionalität ist wichtig, wenn Sie mit blockierenden Einheiten arbeiten, bei denen die Anforderung nach einer gewissen Zeit abgebrochen werden soll. Das kommt beispielsweise in der Netzwerkprogrammierung vor. Das eintreffende Signal unterbricht die blockierende I/O-Funktionen.

```
#include <unistd.h>
long alarm(long Sekunden);
```

Der Rückgabewert ist 0. Steht allerdings noch ein Signal von einem vorher aufgesetzten Alarm aus, wird dieser Alarm gelöscht und die Anzahl der Sekunden zurückgegeben, die noch bis zum Alarm verblieben wären.

[« Auf Signale warten: pause](#) | [Signale](#) | [Zombies vereiteln](#) »

Zombies vereiteln

Einen Zombie könnte man als einen Prozessrest bezeichnen. Genauer gesagt handelt es sich um einen Eintrag in der Prozesstabelle, hinter dem kein echter Prozess mehr steckt. Normalerweise wartet der Vaterprozess auf das Ende des Sohnprozesses. Erst dann läuft er weiter. Dies erreicht der Vaterprozess, indem er `wait()` aufruft und damit in die Warteschlange gesetzt wird. Der Vater kommt wieder frei, wenn der Sohn beendet wurde. Der Rückgabewert des Aufrufs `wait()` liefert dann den Exitstatus des Sohnes. Diesen hinterlegt der Sohn bei seinem Ende in der Prozesstabelle. Wenn aber der Vater gar kein `wait()` ausführt, wird der Eintrag in der Prozesstabelle nie gelöscht. Diesen Eintrag in der Prozesstabelle bezeichnet man als Zombie, weil der Prozess, den er scheinbar anzeigt, gar nicht mehr existiert.

Nun gibt es oft Situationen, in denen nicht gewartet wird, bis der Sohn endet, sondern eben gerade die Parallelität von Prozessen genutzt werden soll. Das heißt, dass besonders bei Dämonen und Serverprozessen immer die Gefahr besteht, dass Zombies entstehen. Um dies zu vermeiden, können Sie das Signal `SIG_CLD` ignorieren. Das passiert, wenn der Funktion `signal()` statt der Behandlungsfunktion die Konstante `SIG_IGN` angegeben wird. Dies sagt dem Betriebssystem, dass dieses Signal in Zukunft nicht beachtet werden soll.

```
signal(SIGCLD, SIG_IGN);
```

Damit teilt der Vaterprozess mit, dass er keineswegs am weiteren Dasein seines Sohnes interessiert ist, und dass bitte keine Nachrichten für ihn aufgehoben werden sollen.

« [Timeout setzen: alarm](#) | [Signale](#) | [Pipe](#) »

Pipe

Unter UNIX ist die Pipe (engl. Röhre) ein Standardmittel zur Verknüpfung zweier Prozesse. Dabei führt der eine Prozess seine Standardausgabe dem Nachfolger als Standardeingabe zu.

Unterabschnitte

- [Prozesskommunikation per Pipe](#)
 - [Named Pipe oder FIFO](#)
 - [Drucken unter UNIX](#)
-

Prozesskommunikation per Pipe

Eine Pipe muss man sich also wie eine Datei mit zwei Enden vorstellen, die eine Richtung hat. Der eine Prozess kann nur schreiben, während der andere nur liest. Der Prozess, der Daten produziert, wirft diese auf der einen Seite in die Pipe hinein. Der andere Prozess konsumiert die Daten, indem er sie der Pipe auf der anderen Seite entnimmt.

```
#include <unistd.h>
int pipe(int zipfel[2]);
```

Es werden mit dem Aufruf von `pipe` zwei Dateihandles erzeugt, die Sie nach dem Aufruf in dem Parameter `zipfel` finden. `zipfel[0]` enthält das Ende zum Lesen und `zipfel[1]` enthält das Ende zum Schreiben in die Pipe. Als nächstes erfolgt typischerweise ein Aufruf von `fork()`. Dann übernimmt der Vater das eine und das Kind das andere Ende der Pipe und schließt das jeweils andere, je nach dem, wer von beiden Datenproduzent und wer Datenkonsument ist. Der Produzent benutzt `write()` und der Konsument `read()`. Beide Enden der Pipe müssen separat per `close()` geschlossen werden.

« [Pipe](#) | [Pipe](#) | [Named Pipe](#) oder [FIFO](#) »

Named Pipe oder FIFO

Unter UNIX können Sie auch eine FIFO-Datei anlegen. Man bezeichnet sie auch als »named pipe«, da sie über ihren Dateinamen angesprochen werden kann. FIFO ist die Abkürzung für First In First Out, lapidar übersetzt mit »wer zuerst kommt, mahlt zuerst«. Alle Arten von Puffern sind FIFOs. Auch die FIFO hat einen Datenproduzenten, der nur in den FIFO hineinschreibt und einen Konsumenten, der nur liest.

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo ( const char *dateiname, mode_t modus);
```

Der Parameter modus nimmt die Berechtigungen auf, wie sie von chmod bekannt sind.

Drucken unter UNIX

Während bei Personal-Computern für das Drucken eine Pseudodatei eröffnet wird und dort hineingeschrieben wird, ist dieser Ansatz unter UNIX nicht realisierbar. Zwar gibt es auch hier eine Datei, die dem Drucker zugeordnet ist (/dev/lp), aber diese ist aus gutem Grund den normalen Benutzern nicht zugänglich. Man möchte vermeiden, dass ein Benutzer einem anderen in den Ausdruck hineinrauscht. Als UNIX-Anwender weiß man, dass man zum Drucken eine Pipe auf das Programm lp oder lpr aufbauen muss. Genauso gehen Sie in einem Programm vor: Sie eröffnen eine Pipe.

```
[Druckerausgabe]
print(char *Inhalt)
{
FILE *PipeID;
```

```
PipeID = popen("lpr", "w");
    fwrite(Inhalt, strlen(Inhalt), 1, PipeID);
    pclose(PipeID);
}
```

[« Named Pipe oder FIFO | Pipe | Fehlerbehandlung mit syslog »](#)

Fehlerbehandlung mit syslog

Unter UNIX gibt es den syslog-Dämon, der Fehlermeldungen des Systems zentral entgegen nimmt und der weitgehend konfigurierbar ist. Als Programmierer ist man herzlich eingeladen, sich an dieser Art der Fehlermeldung zu beteiligen. Das Fehlersystem hat eine relativ simple Schnittstelle und ermöglicht Fehlermeldungen nach Schwere zu klassifizieren. Neben der Arbeitserleichterung, nicht selbst ein Protokoll programmieren zu müssen, hat es den Vorteil, dass die Protokolle dort erscheinen, wo sie auch gefunden werden. Dazu kommt, dass Sie sich nicht um die Beseitigung ihrer alten Protokolldateien kümmern müssen. Vor allem ist es von ungeheurem Vorteil, wenn Sie die Debugstufe des Programmes im laufenden Betrieb herauf- und herabsetzen können. Die folgenden drei Aufrufe müssen in Ihrem Programm vorkommen und schon verteilt der syslog-Dämon Ihre Fehlermeldungen:

```
[Fehlerprotokoll über syslogd]
#include <syslog.h>

...
openlog("MeinProgramm", 0, 0);
...
syslog(LOG_INFO | LOG_LOCAL2,
       "Fehler Nr %d: %s", FehlerNr, Message);
...
closelog();
```

`openlog()` meldet den Programmnamen an, damit er in den Protokollen angezeigt wird. `syslog()` setzt die eigentliche Meldung ab. Der Aufruf entspricht dem der Funktion `printf()`. Er wird lediglich eine Priorität-Konstante vorangestellt. Tabelle zeigt die Prioritäten.

[Fehlerpriorität]L|L Konstante & Bedeutung
LOG_EMERG & Notfall: System ist nicht mehr lauffähig.
LOG_ALERT & Alarm: Eingreifen ist erforderlich
LOG_CRIT & Kritisch: Meist Hardware-Probleme
LOG_ERR & Fehlersituation
LOG_WARNING & Warnung: Es kann aber weitergehen
LOG_NOTICE & Eine wichtige Information
LOG_INFO & Informationen
LOG_DEBUG & Debug-Informationen zur Verfolgung des Prozessablaufes

Dieser Wert muss noch mit der Facility per Oder, also dem senkrechten Strich, verknüpft werden. Mit Facility wird der Auslöser des Fehlers beschrieben. Typische Werte finden Sie in Tabelle .

[Fehlerquelle oder Facility]L|L|L syslog() & syslog.conf & Verursacher
LOG_AUTH & auth & Authorisierung
LOG_AUTHPRIV & auth-priv & Authorisierung
LOG_CRON & cron & Meldungen von cron und at
LOG_DAEMON & daemon & System-Daemonen
LOG_KERN & kern & Kernel
LOG_LOCAL0 bis LOG_LOCAL7 & local0 bis local7 & für eigene Verwendung
LOG_LPR & lpr & Druck Subsystem

LOG_CRIT & & meist Hardware-Probleme
LOG_MAIL & mail &
LOG_NEWS & news &
LOG_SYSLOG & syslog & Nachrichten von syslog selbst
LOG_USER & user & User-Level Nachrichten
LOG_UUCP & uucp & vom uucp

In der /etc/syslog.conf kann festgelegt werden, welche Meldungen in welcher Datei vom syslog-Dämon protokolliert werden soll. Zur Konfiguration des syslog-Dämons siehe S. syslog.

« [Drucken unter UNIX](#) | [UNIX-Systemaufrufe](#) | [Zeitfunktionen](#) »

Zeitfunktionen

UNIX zählt die Zeit in Sekunden seit dem 1.1.1970. Diesen Wert liefert die Funktion `time()`.

```
#include <time.h>
time_t time(time_t *t);
```

Als Parameter können Sie einfach 0 angeben. Der Rückgabewert ist zu einem normalen Integerwert kompatibel. Da die wenigsten Menschen mit der Zahl der Sekunden seit dem 1.1.1970 umgehen können, gibt es hilfreiche Umrechnungsfunktionen.

```
#include <time.h>
struct tm *localtime(const time_t *timer);
```

Die Struktur `tm`, deren Zeiger die Funktion `localtime()` liefert, enthält alle gewünschten Daten, wie das Tagesdatum oder den Wochentag. Alle Werte beginnen bei 0 außer dem Monat. Dieser beginnt mit 1. Auf diese Besonderheit ist vermutlich jeder Programmierer mindestens einmal in seinem Leben hereingefallen. Sie noch nicht? Warten Sie es ab!

```
struct tm {
    int tm_sec;      /* seconds after the minute - [0,61] */
    int tm_min;      /* minutes after the hour - [0,59] */
    int tm_hour;      /* hours - [0,23] */
    int tm_mday;      /* day of month - [1,31] */
    int tm_mon;       /* month of year - [0,11] */
    int tm_year;       /* years since 1900 */
    int tm_wday;       /* days since Sunday - [0,6] */
    int tm_yday;       /* days since January 1 - [0,365] */
    int tm_isdst;      /* daylight savings time flag */
}
```

Um in Protokolldateien einen Zeitstempel zu hinterlassen, gibt es die recht praktische Funktion `asctime()`. Sie liefert anhand der Struktur `tm` eine Zeichenkette, leider im amerikanischen Format. Die Zeichenkette ist immer 26 Byte lang und endet mit einem Zeilenvorschubzeichen und einer abschließenden Null.

```
#include <time.h>
char * asctime(const struct tm *t);
```

Die Funktion `gettimeofday()` ist hilfreich, wenn Sie eine genauere Zeitauflösung als Sekunden brauchen. Sie liefert in seiner Struktur `timeval` auch Mikrosekunden. Wie exakt diese Messung tatsächlich ist, hängt allerdings von der Hardware ab.

```
#include <sys/time.h>
int gettimeofday(struct timeval *zeit, void *tzp);
```

Die Datenstruktur des ersten Parameters enthält die Sekunden und Mikrosekunden und ist folgendermaßen definiert:

```
struct timeval {
    unsigned long    tv_sec;    /* seconds since Jan. 1, 1970 */
    long            tv_usec;    /* and microseconds */
};
```

Der zweite Parameter war ursprünglich für die Zeitzone vorgesehen. Allerdings hat die Umsetzung der verschiedenen Sommerzeitregeln als nicht so einfach erwiesen, so dass Sie als zweiten Parameter am besten 0 angeben.

Mit der Funktion `clock()` werden die Ticks seit dem Start des Programms ermittelt. Um auf Sekunden zu kommen, muss der Wert durch die Konstante `CLOCKS_PER_SEC` geteilt werden. Diese Funktion eignet sich vor allem für Performancemessungen.

```
#include <time.h>
clock_t clock(void);
```

« [Fehlerbehandlung mit syslog](#) | [UNIX-Systemaufrufe](#) | [Benutzer und Gruppen](#) »

Benutzer und Gruppen

Ein Prozess kann erfragen, welcher Benutzer ihn gestartet hat. Der Aufruf `getuid()` liefert die User-ID. Der Aufruf `geteuid()` liefert die effektive User-ID. Diese kann anders lauten, wenn die Programmdatei das SetUID-Bit gesetzt hat. Vergleichbares gibt es für die Gruppe. Dort heißen die Aufrufe `getgid()` und `getegid()`.

```
#include <unistd.h>
uid_t getuid(void);
uid_t geteuid(void);
gid_t getgid(void);
gid_t getegid(void);
```

Unterabschnitte

- [Die Passwortdatei als Struktur](#)
 - [Auslesen der Passwortdatei](#)
 - [Gruppen](#)
-

Die Passwortdatei als Struktur

Soll aus der User-ID der Loginname ermittelt werden, fragt man bei der zentralen Informationsquelle über Benutzer an. Das ist normalerweise die Datei `/etc/passwd`. Dass bei Verwendung von NIS (siehe S. nis) die Information anhand der Daten des NIS-Servers beschafft werden, bemerkt das Programm nicht. Jede Zeile der Passwortdatei wird in einer Struktur namens `passwd` abgebildet.

```
struct passwd {
    char *pw_name; /* Benutzername */
    char *pw_passwd; /* Passwort */
    uid_t pw_uid; /* User ID */
    gid_t pw_gid; /* Group ID */
    char *pw_gecos; /* Name bzw. GECOS */
    char *pw_dir; /* Home-Verzeichnis */
    char *pw_shell; /* Loginshell */
};
```

Ist die User-ID, beispielsweise durch Aufruf von `getuid()`, oder der Loginname bekannt, können Sie den entsprechenden Eintrag mit der Funktion `getpwuid()` oder `getpwnam()` holen. Der Rückgabewert ist ein Zeiger auf eine interne Struktur, die beim nächsten Aufruf überschrieben wird.

```
#include <pwd.h>
#include <sys/types.h>
struct passwd *getpwnam(const char * name);
struct passwd *getpwuid(uid_t uid);
```

Auslesen der Passwortdatei

Um alle Einträge der Passwortdatei auszulesen, gibt es die drei Funktionen `setpwent()`, `getpwent()` und `endpwent()`.

```
#include <pwd.h>
#include <sys/types.h>
struct passwd *getpwent(void);
void setpwent(void);
void endpwent(void);
```

Mit `getpwent()` wird eine Zeile der Passwortdatei nach der anderen eingelesen. Die Funktion `setpwent()` öffnet die Datei und setzt den internen Zeiger auf den Anfang und `endpwent()` schließt die Datei wieder. Um alle Loginnamen der Datei auszulesen, brauchen Sie nur ein recht simples Programm.

```
[Auslesen der Loginnamen]
#include <pwd.h>
```

```
int main()
{
    struct passwd *pw;
```

```
    setpwent();
    while ( pw = getpwent() ) {
        puts(pw->pw_name);
    }
    endpwent();
}
```

Gruppen

Analog funktioniert das Auslesen der Gruppendatei. Auch hier gibt es eine Struktur, die nach der Datei benannt ist.

```
struct group {
    char    *gr_name;    /* Gruppenname */
    char    *gr_passwd;  /* Gruppenpasswort */
    gid_t    gr_gid;     /* Gruppen-id */
    char    **gr_mem;    /* Gruppenmitglieder */
};
```

Die Gruppenmitglieder befinden sich im Strukturelement `gr_mem`. Das ist ein Array von Strings. Das letzte Element des Arrays hat den Wert `NULL`. Auch hier gibt es Funktionen um anhand der ID oder des Namens den entsprechenden Eintrag zu finden. Sie heißen `getgrnam()` und `getgrgid()`.

```
#include <grp.h>
#include <sys/types.h>
struct group *getgrnam(const char *name);
struct group *getgrgid(gid_t gid);
```

Auch das Auslesen der `group`-Datei läuft auf diese Weise:

```
#include <grp.h>
#include <sys/types.h>
struct group *getgrent(void);
void setgrent(void);
void endgrent(void);
```

Das folgende Beispiel liest die `/etc/group` aus und listet für jede Gruppe die Mitglieder auf.

```
[Auslesen der Gruppen und ihrer Mitglieder]
#include <grp.h>
```

```
int main()
{
    struct group *grp;
    char *member;
    int i;

    setgrent();
    while ( grp = getgrent() ) {
        puts(grp->gr_name);
        i = 0;
        member = grp->gr_mem[i++];
        while (member) {
            puts(member);
            member = grp->gr_mem[i++];
        }
        puts("-----");
    }
```

```
}  
    endgrent();  
}
```

Ein Prozess kann mit der Funktion `getgroups()` ermitteln, in welchen Gruppen er angemeldet ist. Dazu bekommt er als Antwort ein Array mit den Group-IDs. Im Beispiel sollen mit Hilfe der Funktion `getgrgid()` die Nummern der Gruppen, zu denen der Aufrufer gehört und dann der zugehörige Name der Gruppe festgestellt werden und so eine Liste auf dem Bildschirm ausgegeben werden.

```
[mygroups]  
#include <grp.h>  
#include <limits.h>  
#include <sys/types.h>
```

```
int main()  
{  
    gid_t grplist[NGROUPS_MAX];  
    struct group *grp;  
    char *member;  
    int i, anzGruppen;
```

```
    anzGruppen = getgroups(NGROUPS_MAX, grplist);  
    for (i=0; i<anzGruppen; i++) {  
        grp = getgrgid(grplist[i]);  
        puts(grp->gr_name);  
    }  
}
```

« [Auslesen der Passwortdatei](#) | **Benutzer und Gruppen** | [Grundlagen der Dämonisierung](#) »

Grundlagen der Dämonisierung

Ein Dämon ist ein Prozess, der im Hintergrund auf Ereignisse wartet und die Bearbeitung dieser Ereignisse übernimmt. Um einen eigenen Dämon zu programmieren, gibt es folgende Teilschritte:

1. Der Prozess wechselt in den Hintergrund. Dazu generiert er von sich selbst einen Kindprozess und beendet sich selbst. Durch diesen Selbstmord wird der Kindprozess zum Waisen und wird vom init-Prozess adoptiert. Daraufhin ist seine PPID 1 (siehe S. daemongeburt).
2. Aufruf von `setsid()`. Damit steigt der Dämon zum Sitzungsführer und zum Prozessgruppenführer auf. Gleichzeitig verliert er sein Kontrollterminal (siehe S. prozessgruppe).
3. Wechsel in das Wurzelverzeichnis. Damit wird verhindert, dass der Prozess ein gemountetes Laufwerk blockiert, wenn es beim Herunterfahren des Systems per `umount` abgehängt werden soll.
4. Schließen nicht mehr benötigter Dateien.

Client-Server Socketprogrammierung

Im Gegensatz zu den anderen Abschnitten der UNIX-Programmierung soll die Socketprogrammierung am Beispiel eines Client-Server-Paares erläutert werden. Dabei soll zunächst die Struktur einer solchen Architektur erläutert werden, und dann die Details betrachtet werden.

Die Socketprogrammierung ist die Grundlage der Programmierung verteilter Anwendungen unter TCP/IP in kommerziellen Client"-Server"-Architekturen als auch bei Internetanwendungen. Ein Socket (engl. Steckdose) ist ein Verbindungsendpunkt, der vom Programm wie eine gewöhnliche Datei mit `read()` und `write()` beschrieben und gelesen werden kann. Ein Socket wird auch mit `close()` geschlossen. Er wird allerdings nicht mit `open()` eröffnet, sondern mit dem Aufruf `socket()`.

Auf der folgenden Abbildung sehen Sie auf der linken Seite den Ablauf eines typischen Servers und auf der rechten einen entsprechenden Client. Die Pfeile dazwischen zeigen auf die Synchronisationspunkte. Die Pfeilrichtung soll zeigen, wer wen freisetzt.

Der Serverprozess muss vom Client eindeutig angesprochen werden können. Dazu bindet er sich mit dem Aufruf `bind()` an eine feste Socketnummer, den so genannten well known port, über den er erreichbar ist. Die Nummer des Ports wird in der Datei `/etc/services` mit einem Servicenamen verbunden. Im Programm kann der Servicenamen durch den Aufruf von `getservbyname()` wieder in eine Zahl umgewandelt werden. Dann bereitet der Server mit `listen()` den Aufruf von `accept()` vor. Der Aufruf von `accept()` blockiert den Prozess bis eine Anfrage von einem Client eintrifft. Direkt anschließend wird der Server `read()` oder alternativ `recv()` aufrufen, um den Inhalt der Anfrage zu lesen. Er verarbeitet die Anfrage und sendet die Antwort an den derzeit wartenden Client. Anschließend kehrt der Server zum `accept()` zurück, um auf weitere Anfragen zu warten.

Der Client braucht keinen festen Port. Er benutzt einen normalen Socket, dem vom System eine freie Nummer zugeteilt wird. Der Server erfährt die Nummer des Clients aus der Anfrage und kann ihm unter diesem Port antworten. Im nächsten Schritt ruft der Client `connect()` auf, um eine Verbindung mit dem Server aufzunehmen, der in den Parametern beschrieben wird. Sobald die Verbindung steht, sendet der Client seine Anfrage per `write()` oder alternativ `send()` und wartet per `read()` oder `recv()` auf die Antwort des Servers. Nach dem Erhalt der Daten schließt der Client seine Verbindung.

Übersicht über die Systemaufrufe

Tabelle fasst die Systemaufrufe, die die Socketprogrammierung betreffen, zusammen.

[Übersicht über die wichtigsten Netzaufrufe]L|L Aufruf & Zweck
socket & Anforderung eines Kommunikationsendpunktes
bind & Lege die Portnummer fest
listen & Festlegen der Pufferzahl für Anfragen
accept & Auf Anfragen warten
connect & Verbindung anfordern
send & Senden von Daten

recv & Empfangen von Daten
close & Schließen des Sockets

In den folgenden Abschnitten werden diese Aufrufe näher untersucht.

Unterabschnitte

- Übersicht über die Systemaufrufe
- Kommunikationsendpunkt: socket und close
- Serveraufrufe: bind, listen und accept
- Clientaufruf: connect
- Datenaustausch: send und recv
- Namensauflösung
- Zahlendreher ntohs und htons
- Rahmenprogramm eines Client-Server Paares
 - Parallelität
- Mehrere Sockets parallel abfragen
- IPv6 aus Programmiersicht
- Client-Server aus Sicht der Performance

« Grundlagen der Dämonisierung | **UNIX-Systemaufrufe** | Kommunikationsendpunkt: socket und close »

Kommunikationsendpunkt: socket und close

Um mit Sockets zu arbeiten, müssen sie zuerst geöffnet werden. Die Funktion `socket()` öffnet einen Socket, die Funktion `close()` schließt ihn. Der Aufruf von `socket()` liefert die Nummer des neuen Socket als Rückgabewert. Falls etwas schiefgelaufen ist, liefert die Funktion -1.

```
#include <sys/socket.h>

int IDMySocket;

IDMySocket = socket(AF_INET, SOCK_STREAM, 0);
...
if (IDMySocket>0) close(IDMySocket);
```

Jeder eröffnete Socket muss auch wieder geschlossen werden. Dies ist an sich eine Binsenweisheit. Eine Nachlässigkeit an dieser Stelle kann sich bitter rächen, da insbesondere bei statuslosen Serverprozessen Verbindungen sehr oft eröffnet werden. Werden sie nicht wieder geschlossen, stehen die Sockets anderen Prozessen nicht mehr zur Verfügung. Im Extremfall kann das Fehlen von Sockets zum Stillstand der kompletten Netzkomponente des Betriebssystems führen.

Das Schließen des Sockets erfolgt unter UNIX mit dem Aufruf `close()`. Dies funktioniert bei anderen Betriebssystemen im Normalfall nicht, da die Analogie der Sockets zu Dateien dort nicht existiert. Meist werden von den TCP/IP-Bibliotheken Namen wie `closesocket`, `socketclose` oder `soclose` verwendet.

Serveraufrufe: bind, listen und accept

Der Serverprozess muss erreichbar sein. Dazu benötigt er einen so genannten well known port. Diese Portnummer ist also den Clientprozessen wohlbekannt. Um einen Socket an diese Nummer zu binden, wird der Aufruf `bind()` verwendet. Als Parameter verwendet er den Socket und eine Struktur `sockaddr_in`, die diesen Port beschreibt. Ist alles in Ordnung, liefert `bind()` als Rückgabewert eine 0, im Fehlerfall -1.

Der Aufruf `listen()` gibt an, wieviele Anfragen gepuffert werden können, während der Server nicht im `accept` steht. In fast allen Programmen wird 5 als Parameter verwendet, da dies das Maximum einiger BSD-Systeme ist. Auch `listen()` liefert als Rückgabewert eine 0, wenn alles glatt lief, ansonsten eine -1.

`accept()` wartet auf die Anfrage eines Clients. Der Aufruf liefert als Rückgabewert den Socket, mit dem der Server im weiteren die Daten mit dem Client austauscht. Er verwendet zum Senden also nicht den gebundenen Socket. Im Fehlerfall liefert `accept()` -1.

```
struct sockaddr_in AdrMySock, AdrPartnerSocket;  
...
```

```
AdrMySock.sin_family = AF_INET;  
AdrMySock.sin_addr.s_addr = INADDR_ANY; /* akzept. jeden */  
AdrMySock.sin_port = PortNr; /* per getservbyname bestimmt */  
bind(IDMySocket, &AdrMySock, sizeof(AdrMySock));  
listen(IDMySocket, 5);  
do {  
    IDPartnerSocket = accept(IDMySocket,  
                             &AdrPartnerSocket, &len);
```

Nicht zu vergessen: `IDPartnerSocket` muss nach Ende der Kommunikation geschlossen werden, obwohl der Socket nicht explizit geöffnet wurde.

Clientaufruf: connect

Sobald der Server gestartet ist, kann der Client Verbindung zum Server aufnehmen. Der entsprechende Aufruf lautet `connect()`. Der Server-Computer wird durch seine IP-Nummer bezeichnet. Diese ist ein 4-Byte-Wert und steht in der `sockaddr_in`-Struktur im Element `sin_addr`. Man erhält diese Nummer durch einen Aufruf von `gethostbyname()`. Der zweite Bestandteil der Serveradresse ist der Zielport. Diesen ermitteln Sie durch den Aufruf von `getservbyname()`. Die Umwandlung von Namen in Nummern wird später näher erläutert.

```
struct sockaddr_in AdrSock;  
  
AdrSock.sin_addr = HostID;  
AdrSock.sin_port = PortNr;  
connect(IDSocket, (struct sockaddr*)&AdrSock, sizeof(AdrSock));
```

Der dritte Parameter gibt die Größe des Speichers an, auf den der Zeiger `AdrSock` zeigt. Dort muss die Größe der Struktur `sockaddr_in` angegeben werden. `connect()` liefert als Rückgabewert 0, wenn alles funktioniert und im Fehlerfall -1.

Datenaustausch: send und recv

Mit den beiden Aufrufen `send()` und `recv()` werden Daten über die bestehenden Verbindungen transportiert. Unter UNIX können dafür auch die Dateiaufrufe `read()` und `write()` verwendet werden. Sofern nicht mit Dateien als auch mit Sockets gearbeitet werden soll, empfiehlt es sich aber, bei `send()` und `recv()` zu bleiben. Erstens erkennt man leichter, dass es sich um Netzverbindungen handelt, zweitens haben Sie Vorteile beim Portieren auf andere Plattformen. Dort arbeiten `read()` und `write()` ausschließlich auf Dateien. Um anstelle von `read()` `recv()` zu verwenden, müssen Sie lediglich den Parameter 0 anhängen. Analoges gilt für `write()` und `send()`.

Die Funktion `recv()` liefert als Rückgabewert die Größe des empfangenen Speicherbereichs oder -1 im Fehlerfall. Da der Rückgabewert nichts über die Größe des tatsächlich gesendeten Pakets aussagt, muss dies vom Programm geregelt werden. Wenn die Pakete nicht immer gleicher Größe sind oder Endezeichen verwendet werden, wird man meist die Paketlänge in den ersten Bytes des ersten Paketes kodieren. Damit weiß der Empfänger, auf wieviel Byte er noch warten muss.

Namensauflösung

Computer und Dienste werden unter TCP/IP eigentlich mit Nummern angesprochen. Allerdings gibt es jeweils Mechanismen zur Namensauflösung. Damit sie auch im Programm Anwendung finden, ruft man die Funktionen `gethostbyname()` zur Ermittlung einer IP-Adresse anhand des Hostnamen und `getservbyname()` zur Ermittlung der Servicenummer anhand des Servicenamens auf.

```
struct hostent *RechnerID;  
struct servent *Service;  
  
/* Bestimme den Rechner namens server */  
RechnerID = gethostbyname("server");  
/* Bestimme den Port für hilfe */  
Service = getservbyname("hilfe","tcp");
```

`gethostbyname()` erhält als Parameter einfach die Zeichenkette mit dem Namen des gesuchten Servers und liefert die IP-Nummer als einen Zeiger auf eine Struktur vom Typ `hostent`. Das wichtigste Element der `hostent`-Struktur ist das Feld `h_addr_list`. Hierin befindet sich das Array der IP-Nummern des Rechners. Das Element `h_addr` liefert die Nummer, wie sie von `connect()` als IP-Nummer gebraucht wird. Dazu muss sie allerdings in das Feld `sin_addr` der Struktur `sock_addr_in` kopiert werden. Das Feld `h_length` liefert die Größe einer IP-Nummer, die bei IPv4 immer vier ist.

```
struct sockaddr_in AddrSock;  
struct hostent *RechnerID;  
  
/* Bestimme den Zielrechner */  
RechnerID = gethostbyname("server");  
bcopy(RechnerID->h_addr,  
      &AddrSock.sin_addr, RechnerID->h_length);  
  
connect(IDSocket, (struct sockaddr *)&AddrSock,  
        sizeof(AddrSock));
```

Die Funktion `getservbyname()` liefert für die beiden Zeichenketten, die den Dienst beschreiben, einen Zeiger auf eine Struktur namens `servent`. Das wichtigste Element der `servent`-Struktur ist das Feld `s_port`. Hierin befindet sich die Nummer des Ports, wie sie von der Funktion `connect()` verwendet wird.

```
struct sockaddr_in AddrSock;  
struct servent *Service;  
  
Service = getservbyname("hilfe","tcp");  
AddrSock.sin_port = Service->s_port;
```

Zahlendreher ntohs und htons

Die Bytefolge ist auf den verschiedenen Computern unterschiedlich definiert. So besteht eine Variable vom Typ short aus zwei Byte. Auf einer Maschine mit Intel-CPU kommt dabei das niederwertige Byte zuerst, während es auf einer Motorola 68000 CPU genau umgekehrt ist. Beispielsweise entspricht die dezimale Zahl 9220 der hexadezimalen 0x2404. Die Inteldarstellung lautet dann 0424, während die Motorola CPU die Zahl als 2404 darstellt. In einem heterogenen Netz muss es dafür einen Standard geben. Unter TCP/IP steht das höherwertige Byte zuerst. Man nennt diese Reihenfolge Big Endian. Der Legende nach stammt diese Bezeichnung aus dem Buch »Gullivers Reisen«, in dem sich zwei Völker darüber zerstreiten, ob man das Ei am dicken Ende (big end) oder am dünnen Ende (little end) zuerst aufmacht. Um die Zahlendarstellung der Maschine in die Netzform zu überführen und die Programme portabel zu halten, gibt es die Makros ntohs() (Net to Host) und htons() (Host to Net). Beide wirken auf short-Variablen. Um long-Variablen zu bearbeiten, gibt es die analogen Makros htonl() und ntohl().

Um beispielsweise den Port des POP3 (110) in die sock_addr_in-Struktur zu schreiben, würde man htons verwenden. Wird an dieser Stelle getservbyname verwendet, erledigt sich die Notwendigkeit von htons.

```
struct sockaddr_in AddrSock;  
    AddrSock.sin_port = htons(110);
```

Unterabschnitte

- Parallelität

Rahmenprogramm eines Client-Server Paares

Ein Server beantwortet in einer Endlosschleife Clientanfragen. Bevor er in diese Endlosschleife geht, muss er seinen Dienst anmelden. Er blockiert erstmals bei `accept()`, der durch den `connect()`-Aufruf des Clients freigegeben wird. Der anschließende Aufruf von `recv()` führt zwar auch zum Blockieren des Servers, aber da der Client sofort seine Anfrage senden wird, ist dies nur von kurzer Dauer. Er sendet die Antwort und wendet sich dem nächsten Anfrager zu.

```
[tcp-Server]
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <netdb.h>

#define MAXPUF 1023

main()
{
    int IDMySocket, IDPartnerSocket;
    struct sockaddr_in AdrMySock, AdrPartnerSocket;
    struct servent *Service;
    int AdrLen;

    char Puffer[MAXPUF];
    int MsgLen;

    IDMySocket = socket(AF_INET, SOCK_STREAM, 0);
    /* Socket an Port-Nummer binden */
    AdrMySock.sin_family = AF_INET;
    AdrMySock.sin_addr.s_addr = INADDR_ANY; /* akzept. jeden */

    /* Bestimme Port */
    Service = getservbyname("hilfe","tcp");
    AdrMySock.sin_port = Service->s_port;

    bind(IDMySocket, &AdrMySock, sizeof(AdrMySock));
    listen(IDMySocket, 5);
    do {
        IDPartnerSocket = accept(IDMySocket,
                                &AdrPartnerSocket, &AdrLen);
        MsgLen = recv(IDPartnerSocket, Puffer, MAXPUF, 0);

        /* tu was mit den Daten */
```

```

send(IDPartnerSocket, Puffer, MsgLen, 0);
close(IDPartnerSocket);
} while(1); /* bis zum St. Nimmerlein */
}

```

Dieser Server bearbeitet nacheinander jede Anfrage, die über den Port »hilfe« an ihn gestellt wird. Nach jeder Anfrage wird die Verbindung wieder gelöst und ein anderer Client kann anfragen. Ein solcher Server dürfte auf jedem Betriebssystem arbeiten, das TCP/IP unterstützt. Lediglich der Aufruf von `close()` muss angepasst werden.

Der zugehörige Client bereitet die Verbindung in der Variablen `AdrSocket` vor und ruft damit die Funktion `connect()` auf. Diese blockiert bis der Server auf der anderen Seite `accept()` aufgerufen hat. Der Client fährt fort, indem er seine Anfrage sendet. Der Sendevorgang blockiert nie, dafür aber der anschließende Empfang der Antwort. Sobald der Server seine Antwort gesendet hat, kann der Client sich beenden.

```

[tcp-Client]
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <netdb.h>

#define MAXPUF 1023

main()
{
    int IDSocket;
    struct sockaddr_in AdrSock;
    int len; /* Die Laenge der Socketstruktur */

    struct hostent *RechnerID;
    struct servent *Service;
    char Puffer[MAXPUF];

    IDSocket = socket(AF_INET, SOCK_STREAM, 0);

    /* Bestimme den Zielrechner */
    RechnerID = gethostbyname("server");
    bcopy(RechnerID->h_addr,
          &AdrSock.sin_addr, RechnerID->h_length);

    /* Bestimme den Port */
    Service = getservbyname("hilfe","tcp");
    AdrSock.sin_port = Service->s_port;

    connect(IDSocket, (struct sockaddr *)&AdrSock,
            sizeof(AdrSock));

    send(IDSocket, Puffer, MAXPUF, 0);
    recv(IDSocket, Puffer, MAXPUF, 0);
    close(IDSocket);
}

```

Es gibt zwei Variablen pro Socket. Die eine ist wie bei Dateizugriffen ein einfaches Handle (hier mit ID gekennzeichnet), die andere enthält die Adresse der Verbindung, also die Internet-Nummer des Rechners und die Nummer des Ports. Der Server legt die IP-Nummer des Rechners nicht fest, von dem er Anfragen akzeptiert. Das erreicht er, indem er die Konstante `INADDR_ANY` benutzt wird. Der Client dagegen gibt die Adresse des anzusprechenden Servers an. Die Funktion `recv()` liefert als Rückgabewert die Größe des versendeten Speicherbereichs. Die Funktion `recv()` liest die Sendung in Paketen von maximal 1KB. Wurden größere Pakete verschickt, müssen sie häppchenweise gelesen werden. Das Senden ist nicht beschränkt.

Parallelität

Der Server wird nun ergänzt, damit er die Vorteile einer Multitaskingumgebung nutzen und mehrere Anfragen parallel abarbeiten kann. Dazu muss an passender Stelle ein `fork()` eingebaut werden:

```
[Multitasking Server]
do {
    IDPartnerSocket = accept(IDMySocket,
                            &AdrPartnerSocket, &len);
    if (fork()==0) {
        MsgLen = recv(IDPartnerSocket, Puffer, MAXPUF, 0);

/* tu was mit den Daten */

send(IDPartnerSocket, Puffer, MsgLen, 0);
close(IDPartnerSocket);
/* Sohn toetet sich selbst */
exit(0);
} /* if fork.. */
close(IDPartnerSocket); /* der Vater schliesst Verbindung */
} while(1);
```

Man sieht, mit welcher geringer Änderung ein multitaskingfähiger Server zu realisieren ist. Beim Aufruf von `fork()` wird von dem Prozess ein Kindprozess erzeugt, der alle Ressourcen des Vaters besitzt. So kann er die Verbindung mit dem Anfrager weiter bearbeiten. Er tritt vollkommen an die Stelle des Vaters, der seinerseits die Verbindung schließen kann und auf eine neue Anfrage wartet. Sobald diese eintrifft, wird wieder ein Kind generiert, der gegebenenfalls parallel zum anderen Kind arbeitet, falls jenes noch nicht fertig ist.

Der Server ist so, wie er nun vorliegt, ein statusloser Server (stateless server). Das bedeutet, er kann sich den Stand einer Kommunikation nicht merken. Fragt derselbe Client noch einmal an, wird er ihn wie eine völlig neue Anfrage behandeln. In dieser Art arbeitet ein Webserver. Jede Anfrage ist für ihn neu. Andere Server, beispielsweise POP3-Server, halten die Verbindung mit ihrem Client solange aufrecht, bis beide ein Ende der Verbindung vereinbaren. In solch einem Fall würde eine Schleife im Sohnprozess über `recv()`, Verarbeitung und `send()` laufen, bis ein definiertes Ende der Kommunikation stattfindet. Natürlich würde dann auch der Client eine Schleife haben, die erst bei Einigung über den Verbindungsabbau beendet wird.

Wie im Zusammenhang mit Signalen gezeigt, sollte die Entstehung von Zombies verhindert werden. Zombies entstehen, wenn ein Sohn endet, aber der Vater nicht auf ihn wartet. Dadurch bleibt ein Eintrag in der Prozesstabelle mit dem Exitwert des Sohnes. Der Aufwand ist denkbar gering:

```
signal(SIGCLD, SIG_IGN);
```

Diese Zeile sollte dem Server eingebaut werden, bevor er in die Endlosschleife läuft.

« [Zahlendreher ntohs und htons](#) | [Client-Server Socketprogrammierung](#) | [Mehrere Sockets parallel abfragen](#)
»

Mehrere Sockets parallel abfragen

Ein anderes Thema der Socketprogrammierung ist die parallele Bearbeitung mehrerer Ports durch einen einzigen Prozess. Ein Beispiel für ein solches Programm ist `inetd`, der so genannte Internetdämon, der für alle in der `inetd.conf` aufgeführten Ports die Anfragen entgegen nimmt und den benötigten Server aufruft. Er arbeitet mit dem Aufruf `select()`, der mehrere Dateideskriptoren und damit Sockets parallel abhören kann.

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
int select(int MaxHandle, fd_set *Lesen,
           fd_set *Schreiben, fd_set *OutOfBand,
           struct timeval *TimeOut);
```

Zentrale Objekte sind in diesem Zusammenhang die Dateideskriptorenarrays. Von diesen nimmt `select()` drei an. Da gibt es je einen Satz zum Lesen, zum Schreiben oder um Ausnahmen, wie Nachrichten außerhalb der Reihe zu beobachten. Wollen Sie nicht alle Kategorien beobachten, geben Sie bei den uninteressanten Parametern den Wert `NULL` an.

Sollen die Sockets `a` und `b` zum Lesen beobachtet werden, dann muss ein `fd_set` angelegt und damit gefüllt werden:

```
fd_set lesesockets;
```

```
FD_ZERO(&lesesockets);
FD_SET(a, &lesesockets);
FD_SET(b, &lesesockets);
```

```
maxHandle = max(a, b) + 1;
```

```
select(maxHandle, &lesesockets, NULL, NULL, NULL);
```

`select()` blockiert so lange, bis auf einem der Sockets Daten ankommen. Soll ein Timeout definiert werden, so muss der letzte Parameter besetzt werden. Dann wird die Funktion nach einer gewissen Zeitspanne unterbrochen, die in einer Variablen von Typ `timeval` festgelegt wird, und als letzter Parameter an `select()` übergeben wird. Im Beispiel wird eine zehntel Sekunde eingestellt.

```
struct timeval myTime;
...
myTime.tv_sec = 0;
myTime.tv_usec = 100000;
```

```
select(maxHandle, &lesesockets, NULL, NULL, &myTime);
```

Die Fähigkeit von `select()`, einen Timeout im Mikrosekundenbereich festzulegen, wird manchmal dazu missbraucht, um einen Prozess für kurze Zeit schlafen zu legen. Die eigentlich dafür zuständige

Funktion `sleep()` legt einen Prozess für mindestens eine Sekunde schlafen, was manchmal zu viel ist. Um dies zu erreichen, kann man die ersten vier Parameter auf 0 setzen.

« [Rahmenprogramm eines Client-Server Paares](#) | [Client-Server Socketprogrammierung](#) | [IPv6 aus Programmiersicht](#) »

IPv6 aus Programmiersicht

Die kommende Generation der IP-Adressen, die nun 16 statt 4 Byte als Adressen verwenden, haben natürlich auch Konsequenzen in der Programmierung. Es wird neue Konstanten, Strukturen und Funktionen geben, die anstelle der alten Versionen eingesetzt werden müssen. An den Konzepten, wie Client- und Serverprogramme strukturiert sind, ändert sich nichts. Insofern sind die Administratoren von den Neuerungen stärker betroffen als die Programmierer.

Es ist die neue Adressfamilie `AF_INET6` statt `AF_INET` definiert. Statt der Struktur `in_addr` gibt es die Struktur `in6_addr`, die wie folgt definiert wird:

```
struct in6_addr {
    uint8_t s6_addr[16];
```

Durch diese Änderung ist dann auch die Struktur `sockaddr_in` betroffen, die nun als `sockaddr_in6` folgendermaßen definiert ist:

```
struct sockaddr_in6 {
    sa_family_t      sin6_family;
    in_port_t        sin6_port;
    uint32_t          sin6_flowinfo;
    struct in6_address sin6_addr;
}
```

Die Umsetzung zwischen Hostname zu IP-Nummer wird nicht mehr mit den Funktionen `gethostbyname()` bzw. `gethostbyaddr()`, sondern durch die neuen Funktionen `getnodebyname()` bzw. `getnodebyaddr()` erledigt. vgl. Santifaller, Michael: TCP/IP und ONC/NFS - Internetworking mit UNIX. Addison-Wesley, Bonn, 1998. S. 365

Client-Server aus Sicht der Performance

Das Ziel einer guten Client-Server-Architektur besteht darin, die Leistung, die im lokalen Rechner ansonsten brachläge, für Dinge zu nutzen, die nicht zentral ablaufen müssen und damit den zentralen Rechner zu entlasten.

Betrachtet man eine Software, die von mehreren Anwendern gleichzeitig benutzt werden soll, ergeben sich drei Architekturmodelle. Die klassische Variante stellt jedem Teilnehmer ein Terminal zur Verfügung und der Zentralrechner führt alle Anforderungen auf seinem zentralen Prozessor und seiner Platte aus. Dazu gehört auch die Benutzerführung, beispielsweise das Aufbauen der Masken. Damit belastet auch das Navigieren im Programm auf der Suche nach der richtigen Maske die Allgemeinheit.

Bei der Lösung mit einem Plattenserver wird die Prozessorlast auf die Arbeitsplätze verteilt. Da der Server keine Eigenintelligenz mitbringen muss, reicht normalerweise ein handelsüblicher PC. Man übersieht allerdings leicht, dass alle Plattenzugriffe auch das Netz belasten, das typischerweise langsamer als der Plattenzugriff ist. Soll beispielsweise nach einem bestimmten Kunden gesucht werden, erfolgt bei einem Sortierten oder indizierten Datenbestand eine binäre Suche. Das bedeutet, der erste Zugriff erfolgt auf den mittleren Datensatz. Ist der gefundene Name im Alphabet höher, so wird die untere Hälfte halbiert, ansonsten die obere Hälfte. Auf diese Weise findet man mit 10 Zugriffen den richtigen Satz in 1024 Sätzen. Für 2048 Sätze braucht man 11 Zugriffe, für 4096 12 und so weiter. Diese Zugriffe laufen aber alle über das Netz und bei einer hohen Netzbelastung wird das Gesamtverhalten in erhebliche Mitleidenschaft gezogen.

Eine Client-Server Lösung kann an einer beliebigen Stelle geteilt werden. Man wird den Teilungspunkt oberhalb der binären Suche ansetzen, so dass die Plattenzugriffe lokal auf dem Server stattfinden und so das Netz nicht belasten. Auf der anderen Seite wird man die Benutzersteuerung auf dem lokalen Arbeitsplatz belassen, so dass nur schlanke Pakete mit Anfragen an den Server gerichtet werden.

Reguläre Ausdrücke

In vielen Programmen werden zum Suchen von Mustern reguläre Ausdrücke verwendet. Beeindruckend ist, dass alle Programme die Ausdrücke gleich interpretieren. Das liegt allerdings nicht in erster Linie an der Harmoniesucht der UNIX-Programmierer, sondern einfach daran, dass UNIX die benötigten Funktionen als Bibliothek zur Verfügung stellt.

Nach POSIX-Standard werden vier Funktionen zur Verfügung gestellt: `regcomp()`, `regexexec()`, `regerror()` und `regfree()`. `regcomp()` übersetzt eine Zeichenkette mit einem regulären Ausdruck in eine Variable vom Typ `regex_t`. Diese wird dann mit der Funktion `regexexec()` auf den zu durchsuchenden Text angewandt. Wird die Variable nicht mehr benötigt, wird sie mit `regfree()` freigegeben. `regerror()` wandelt die Fehlernummern, die `regcomp()` oder `regexexec()` erzeugen, in eine Fehlermeldung um. Programme, die diese Funktionen benutzen müssen die Datei `regex.h` einbinden.

Als Beispiel dient hier eine einfache Nachbildung des Programmes `grep`.

```
[Regulärer Ausdruck in grep]
#include <stdlib.h>
#include <stdio.h>
#include <regex.h>

int main(int argc, char *argv[])
{
    char puffer[512];
    int i;
    regex_t regexpr;
    FILE *fp;

    if (argc<3) {
        printf("Usage: %s regexexpression filesn", argv[0]);
        return -1;
    }
    if (regcomp(&regexpr, argv[1], REG_EXTENDED|REG_NEWLINE)) {
        printf("Problem beim Ausdruck %sn", argv[1]);
        return -2;
    }
    for (i=2; i<argc; i++) {
        fp = fopen(argv[i], "r");
        if (fp!=NULL) {
            while (!feof(fp)) {
                fgets(puffer, sizeof(puffer)-1, fp);
                if (regexexec(&regexpr, puffer, 0, NULL, 0)==0) {
                    puts(puffer);
                }
            }
            fclose(fp);
        }
    }
    regfree(&regexpr);
    return 0;
}
```

Weitere Programmierschnittstellen

In diesem Abschnitt möchte ich noch einige Bibliotheken und Schnittstellen nennen, die unter UNIX verfügbar sind.

System V brachte das Konzept der Streams hervor. Dabei werden Messages mit den Funktionen `putmsg()` und `getmsg()` an einen Stream gesandt. Für Fehlermeldungen stellt System V einen Stream `log` zur Verfügung, der mit dem `syslog`-Dämon zusammenarbeitet.

Oberhalb der Socketprogrammierung gibt es das RPC (Remote Procedure Call), das die Netzwerkkommunikation als Funktionsaufruf modelliert. NFS und NIS basieren auf dem RPC.

Memory Mapped I/O ist sowohl in BSD als auch in System V verfügbar. Die Funktionen lauten `mmap()`, `msync()` und `munmap()`. Eine Datei wird auf einen Speicherbereich abgebildet und kann dann mit Speicherzugriffen behandelt werden. Unter Linux sind damit beispielsweise das dynamische Laden von Programmteilen realisiert worden.

Curses ist eine Bibliothek, um Texte in Terminals zu positionieren. Damit ist es sogar möglich, einfache textuelle Fenster zu erstellen. Mit `ncurses` bietet Linux sogar eine farbige Variante an.

Systemkonformität

UNIX ist eine Multitaskingumgebung. Obwohl das Betriebssystem Prozesse gegeneinander abschirmt, liegt es dennoch in der Verantwortung des Programmierers systemkonform zu programmieren. Was das bedeutet, sei hier an zwei Beispielen erläutert.

Unterabschnitte

- [Polling](#)
 - [Rechte beachten](#)
-

Polling

Polling bedeutet, dass ein Programm in einer Endlosschleife prüft, ob ein Ereignis eintrifft. Dieses Verfahren wurde in Singletaskingsystemen wie MS-DOS oder auf Kleincomputern eingesetzt, um beispielsweise die Tastatur abzufragen oder auch beim Drucken. Beide Fälle spielen im UNIX-Bereich kaum eine Rolle, da der direkte Zugriff auf die Tastatur oder auf den Drucker nicht möglich ist.

Allerdings gibt es immer wieder Aufgabenstellungen, wo ein Programm auf ein Ereignis warten muss, das nicht durch eine Systemschnittstelle abgedeckt ist. Beispielsweise wird eine Programmkommunikation oft über Dateien realisiert und der Konsument wartet, bis eine Datei erzeugt wird, bzw. bis die Sperrdatei gelöscht wird. In so einem Fall muss in die Warteschlange ein Aufruf von `sleep` eingebaut werden. Ansonsten zieht dieser Prozess nach und nach einen Großteil der CPU-Zeit an sich und andere Prozesse stehen im Nebel.

[« Systemkonformität | Systemkonformität | Rechte beachten »](#)

Rechte beachten

Manchmal stehen einem Anwendungsprogramm ein paar Rechte im Wege. Es ist natürlich am einfachsten, wenn man diese fehlenden Zugriffsrechte verändert und den Weg für das Programm frei macht. Im Normalfall sollte eine solche Situation aber auf »legalen« Wegen zu umgehen sein und das Stolpern über mangelnde Rechte ein Hinweis darauf sein, dass hier vielleicht nicht systemkonform programmiert wird.

Ein triviales Beispiel ist der Drucker. Das Device des Druckers liegt bekanntlich unter `/dev/lp` oder `/dev/lp0`, ist aber für den normalen Anwender und sein Programm nicht beschreibbar. Natürlich kann ein Programm dieses Device als Datei öffnen und hineinschreiben. Allerdings wird damit der Spooler umgangen und ein Teil der Ressource ausgeschlossen.

[« Polling | Systemkonformität | unix »](#)

[gpGlossar18133]

-
- [Literatur](#)
-

Literatur

UNIX

Banahan, Mike / Rutter, Andy: UNIX - lernen, verstehen, anwenden. Carl Hanser Verlag, Münschen-Wien, 4. unveränd. Aufl. 1989.

Barkakati, Naba: LINUX Red Hat 6.0. Franzis', Poing, 2000.

Boor, Thomas / Hutter, Joachim / Pribas, Arnulf A.C. Pribas: vi-Referenzhandbuch. Prentice Hall, Haar bei München, 1996.

Comer, Douglas: Operating System Design - The XINU Approach. Prentice Hall International Editions, Englewood Cliffs, New Jersey, 1984.

Detering, Reinhard: UNIX Handbuch System V. Sybex, Düsseldorf-San Franzisko-Paris-London-Soest, 4. Aufl., 1990.

Husain, Kamran/Parker, Timothy, Ph.D., et al.: Red Hat Linux Unleashed. SAMS, Indianapolis, 1996.

Kienle, Michael/Jaeschke, Gerhard: Reisebegleiter. iX 7/1993, S. 26-34.

Loukides, Mike: System Performance Tuning - Optimierung von UNIX Systemen. Addison Wesley, Bonn, 1993.

Milz, Harald: Crashfest - SGI XFS auf SuSE 7.1.. Linux-Magazin 07/2001, S. 86-89.

Nemeth, Evi / Snyder, Garth / Seebass, Scott: UNIX System Administration Handbook. Prentice Hall, Englewood Cliffs, New Jersey, 1989.

Nemeth, Evi / Snyder, Garth / Seebass, Scott / Hein, Trent R.: UNIX Systemverwaltung. Markt+Technik - Prentice Hall, München, 2001.

Schmidt, Fabian: Backup ohne Klicks. Linux Magazin 05/2002. S. 52-54.

Tanenbaum, Andrew S.: Operating Systems - Design and Implementation. Prentice Hall, Englewood Cliffs, New Jersey, 1987.

Thissen, Thomas: Umlaute auf Umwegen. iX 9/1993, S. 194-197.

Tondo, Nathanson, Yount: Das Softwarewerkzeug Make. Prentice Hall, Haar bei München, 1994.

Wobst, Reinhard: "cpio" statt "tar". UNIXmagazin 10/92, S.26-36.

Netzwerke

- Albitz, Paul / Liu, Cricket: DNS und BIND. O'Reilly, Köln, 2. korr. Nachdruck, 2001.
- Bown, Rich / Coar, Ken: Apache und CGI. Markt+Technik, München, 2000.
- Comer, Douglas E.: Computernetzwerke und Internets. Prentice Hall, Pearson, München, 2000.
- Comer, Douglas E.: Internetworking with TCP/IP. Prentice Hall, Englewood Cliffs, 2nd ed., 1991.
- Endres / Schmidt: Bei Anruf Netz. c't 25/99, S. 218-223.
- Ernst, Yves: HTML. Data Becker, Düsseldorf, 1996. Kapitel 13: Common Gateway Interface.
- Hunt, Craig: TCP/IP Network Administration. O'Reilly, Sebastopol, 1994.
- Tanenbaum, Andrew S.: Computer Networks. Prentice Hall, Englewood Cliffs, 1987.
- Röhrig, Bernhard: Linux im Netz. C&L, Vaterstetten, 1997.
- Roscher, Andreas: LINUX als Windows Server. dpunkt, Heidelberg, 2. Aufl. 1998.
- Santifaller, Michael: TCP/IP und ONC/NFS - Internetworking mit UNIX. Addison-Wesley, Bonn, 1998.
- Ziegler, Robert: Linux Firewalls. Markt+Technik, München, 2000.

X Window System

- Eßer, Hans-Georg: KDE Der neue Desktop für Linux. Sybex, Düsseldorf, 1999.
- Mansfield, Niall: Das Benutzerhandbuch zum X Window System. Addison Wesley, Bonn, 1991.
- Mikes, Steven: X Window System Program Design and Development. Addison-Wesley, Reading, 1992.
- OSF/Motif User's Guide. Open Software Foundation. Revision 1.2. 1992.

Programmierung

- Beck, Michael: Linux Kernelprogrammierung. 4. akt. u. erw. Aufl., Addison Wesley, Bonn, 1997.
- Herold, Helmut: Linux-Unix-Systemprogrammierung. 2. überarb. Aufl. Addison Wesley, München, 1999.
- Husain, Kamran: Perl. in Red Hat Linux Unleashed, Chapter 29, Sams Publishing, Indianapolis, 1996, pp. 497-522.

Illik, J. Anton: Programmieren in C unter UNIX. Sybex, Düsseldorf, 1992.

Johnson, Andrew L.: Perl - Der Einstieg. Galileo Computing, Bonn, 2001.

Johnson, Michael K. / Troan, Erik W.: Anwendungen entwickeln unter Linux. Addison Wesley, Bonn, 1998.

Ousterhout, John K.: Tcl und Tk. Addison-Wesley, Bonn, 1995.

Rochkind, Marc J.: UNIX Programmierung für Fortgeschrittene. Hanser Verlag, München-Wien, 1988.

Stevens, W. Richard: Programmierung in der Unix-Umgebung. Addison Wesley, Bonn, 1995.

Wirth, Niklaus: Algorithmen und Datenstrukturen mit Modula-2. 4. Aufl., Teubner, Stuttgart, 1986.

« [unix](#) | [unix](#) |